Universidade de Aveiro
2018

Departamento de Eletrónica, Telecomunicações e Informática

Ana Filipa Vinhas Mendes

Sistema de registo de intervenções clínicas codificadas de acordo com a CIF

Universidade de Aveiro
2018

Departamento de Eletrónica, Telecomunicações e Informática

Ana Filipa Vinhas Mendes

Sistema de registo de intervenções clínicas codificadas de acordo com a CIF

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática, realizada sob a orientação científica do Doutor Joaquim Arnaldo Martins, Professor do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Nelson Fernando Pacheco da Rocha, Professor do Departamento de Ciências Médicas da Universidade de Aveiro.

o júri

presidente	Prof. Doutor José Manuel Matos Moreira professor associado da Universidade de Aveiro
vogais	Prof. Doutor Fernando Joaquim Lopes Moreira professor associado da Universidade Portucalense
	Prof. Doutor Joaquim Arnaldo Martins professor catedrático associado da Universidade de Aveiro

agradecimentos

Dirijo uma palavra de agradecimento sincero a quem me acompanhou ao longo do meu percurso académico, reconhecendo os contributos de entidades e pessoas não só para esta dissertação, mas também para o meu enriquecimento académico e pessoal.

Em primeiro lugar gostaria de agradecer aos meus pais e avós por todo o apoio económico, pela força e pelo carinho que sempre me prestaram ao longo desta etapa, bem como à elaboração da presente dissertação e a qual sem o seu apoio teria sido impossível. Um grande obrigado ao meu irmão e à minha cunhada que disponibilizaram algumas das suas horas de lazer para me ajudar e incentivar na concretização deste objetivo.

Aos meus orientadores, Professor Joaquim Arnaldo Martins e Professor Nelson Fernando Pacheco da Rocha, pela sua disponibilidade e apoio manifestado ao longo da realização desta dissertação.

Ao Pedro por ter caminhado ao meu lado, pela sua paciência, compreensão e ajuda prestada durante a elaboração da presente dissertação, mesmo quando não sabia como auxiliar. Apesar de tudo apresentou sempre um sorriso, quando sacrificava os dias, as noites, os fins-de-semana e os feriados em prol da realização deste estudo.

Agradeço também a todos aqueles que se predispuseram a ajudar-me tanto no conteúdo deste projeto como com a sua presença e uma palavra amiga. Agradeço a vossa atenção e paciência. Para não correr o risco de não enumerar alguém não vou identificar ninguém, aqueles a quem este agradecimento se dirige sabê-lo-ão.

Sem esquecer, queria agradecer ao meu confidente Sr. Zé pelas tardes de café e chocolate onde me ouviu e apoiou nos momentos mais difíceis.

palavras-chave CIF, MEAN *Stack*, aplicação *web*

resumo

A Classificação Internacional de Funcionalidade, Incapacidade e Saúde (CIF), que está introduzida nas classificações desenvolvidas pela Organização Mundial da Saúde, tem sido incorporada e utilizada em diversos setores da saúde e equipas multidisciplinares. Apesar da importância e atualidade da CIF, alguns conceitos desta foram pouco detalhados e justificados, podendo ocasionar interpretações distintas. Este facto torna o processo de codificação (tradução entre as informações recolhidas para os códigos da CIF) uma tarefa bastante complexa, suscetível a erros e que exige um tempo excessivo por parte dos avaliadores, dado que não existe qualquer tipo de ferramenta de suporte informático. As únicas plataformas encontradas não oferecem vantagens no processo de codificação, uma vez que fornecem apenas um motor de busca sobre a CIF.

Com este projeto pretende-se assim dotar os clínicos com um sistema que, com base em regras predefinidas, os auxiliem a codificar e registar as intervenções clínicas e os respetivos resultados de uma forma mais eficiente e menos propensa a erros de subjetividade.

A solução envolveu a recolha e formatação dos dados da CIF na língua portuguesa e o seu armazenamento numa aplicação web desenvolvida com o auxílio do conjunto de tecnologias denominado MEAN *Stack*. Esta aplicação procura revolucionar o processo de codificação principalmente na área da Fisioterapia Ocupacional. Com o armazenamento de catalogações realizadas por profissionais numa plataforma unificada e partilhada, é assim minimizado o risco de perda ou esquecimento de dados. Além disto, através da simplificação da catalogação de intervenções clínicas, apresenta-se como uma vantagem para indivíduos com pouca experiência, uma vez que ao longo da sua utilização o número de exemplos e o nível de detalhe da CIF vai aumentando.

keywords ICF, MEAN *Stack*, web application

abstract

The International Classification of Functioning, Disability and Health (ICF), which is introduced in the classifications developed by the World Health Organization, has been incorporated and used in several health sectors and multidisciplinary teams. Despite its importance and timeliness, some concepts were poorly detailed and its usage limitedly justified, leading to varied interpretations. This fact makes the coding process (translation between information collected for ICF codes) a very complex task, susceptible to errors and requires an excessive time by the evaluators, since there is no type of informatic tool. The only platforms available to use do not currently offer any advantages in terms of the coding process, functioning only as a search engine of ICF.

This project aims to provide clinicians with a system that, based on predefined rules, will help them to code and record clinical interventions and its results in a more efficient way and less prone to subjectivity errors.

The solution involved the collection and parsing of the ICF data in the Portuguese language and its storage in a web application developed with the help of the set of technologies called MEAN *Stack*. This application seeks to revolutionize the linking process mainly in the area of Occupational Physiotherapy. By storing the mapping performed by professionals on a unified and shared platform, the risk of lost or misplaced data is minimized. In addition, simplifying the cataloging of clinical interventions is advantageous for individuals with little experience, since the number of examples and the level of detail in ICF increases during its usage.

Índice

Índice de Ilustrações

Índice de Tabelas

Glossário

AJAX	Asynchronous JavaScript and Extensible Markup Language
API	Application Programming *Interface*
BSON	Binary JavaScript Object Notation
CID	Classificação Internacional de Doenças e Problemas Relacionados à Saúde
CIF	Classificação Internacional de Funcionalidade, Incapacidade e. Saúde
CORS	Cross-Origin Resource Sharing
CRLF	Carriage Return, Line Feed
CRUD	Create, Read, Update e Delete
CSS	Cascading Style Sheets
DNS	Domain Name System
DOM	Document Object Model
EJS	Embedded JavaScript templating
ES	ECMAScript
FTP	File Transfer Protocol
FURPS	Functionality, Usability, Reliability, Performance and Supportability
GB	Gigabyte
GridFS	Grid file system
GUI	Graphics User *Interface*
HATEAOS	Hypermedia as the Engine of Application State
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
I/O	Input/Output
ICIDH	International Classification of Impairments, Disabilities, and Handicaps
IDE	Integrated Development Environment
INR	Instituto Nacional para a Reabilitação
IoT	Internet of Things
IP	Internet Protocol
Java EE	Java Platform, Enterprise Edition
Java ME	Java Plataform, Micro Edition

Java SE	Java Platform, Standard Edition
JSON	JavaScript Object Notation
JSP	Java Server Pages
JSX	JavaScript Extensible Markup Language
JWT	JavaScript Object Notation Web Token
MB	Megabyte
MEAN	MongoDB - Express - AngularJS - NodeJS
MIT	Massachusetts Institute of Technology
MVC	Model-view-controller
NoSQL	Not Only Structured Query Language
NPM	Node Package Modules
ODM	Object Document Mapper
OMS	Organização Mundial da Saúde
ORM	Object-relational mapping
PCCIF	Plataforma de Codificação da Classificação Internacional da Funcionalidade, Incapacidade e Saúde
PEM-encoded	Privacy Enhanced Mail enconded
PHP	Hypertext Preprocessor
POO	Programação Orientada a Objetos
REST	Representational State Transfer
RFC	Request For Comments
SGBD	Sistema de Gestão de Banco de Dados
SGBDR	Sistema de Gestão de Banco de Dados Relacional
SMTP	Simple Mail Transfer Protocol
SNOMED CT	Systematized Nomenclature of Medicine Clinical Terms
SPA	Single-Page Application
SQL	Structured Query Language
TCP/IP	Transmission Control Protocol / Internet Protocol
UI	User *Interface*
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VS Code	Visual Studio Code
WHODAS	World Health Organisation Disability Assessment Schedule
XML	Extensible Markup Language

CAPÍTULO 1

Introdução

1.1 Motivação

Como agência especializada em saúde, a Organização Mundial da Saúde[1] (OMS) tem a responsabilidade de trabalhar a harmonização da linguagem médica. Como tal, devido às mudanças paradigmáticas na área da saúde e à falta de uma definição clara dos conceitos deficiência e incapacidade, a qual tem sido apresentada como um impedimento para a promoção da saúde de pessoas com deficiência, sustentou a necessidade da supracitada entidade, desenvolver indicadores de saúde mais direcionados às consequências das doenças, ao invés da mortalidade e morbilidade [1]. Desta forma, surge a Classificação Internacional de Funcionalidade, Incapacidade e Saúde[2] (CIF) que, como o próprio nome indica, se propõe a retratar os aspetos da funcionalidade, incapacidade e saúde das pessoas, fundamentada numa abordagem biopsicossocial da saúde. Esta classificação assume que a funcionalidade de um indivíduo com uma determinada condição de saúde depende de aspetos físicos, pessoais e ambientais [2].

A classificação é o processo de transformar descrições de diagnósticos e procedimentos em códigos universais. A CIF é uma classificação hierárquica e utiliza um sistema alfanumérico no qual as letras **b**, **s**, **d** e **e** são utilizadas para indicar os quatro domínios no qual esta se subdivide, respetivamente, funções do corpo, estruturas do corpo, atividades e participação e fatores ambientais. Cada domínio está organizado por capítulos, onde as letras são seguidas por o número do capítulo (dígito). Os dois dígitos seguintes representam o segundo nível, seguido pelo terceiro nível (um dígito) e por último, o quarto nível (um dígito) [2]. Esta organização é possível ser analisada na Tabela 1.

[1] http://www.who.int/
[2] http://www.who.int/classifications/icf/en/

Tabela 1 - Exemplo de uma codificação da CIF [3] .

Código	Domínio	Capítulo	2º Nível	3º Nível	Qualificador
b7302.1	b - Funções do Corpo	7 - Capítulo 7: Funções neuro-músculo-esqueléticas e relacionadas com o movimento	30 - Funções de força muscular	2 - Força dos músculos de um lado do corpo	1 - Deficiência leve

Segundo a OMS, os códigos da CIF só se encontram completos após a presença de um qualificador, que indica a magnitude do nível de saúde. Podem ser codificados com um, dois ou mais dígitos após um ponto separador [2].

Os níveis representam o nível de detalhe da classificação, isto é, as categorias mais amplas podem conter subcategorias mais detalhadas. Por exemplo, o capítulo Mobilidade (d4), encontra-se incluído dentro no domínio "Atividades e Participação" (d) e integra subcategorias como: andar (d450-d469), sentar (d410-d429) e transportar objetos (d430-d449). É possível atribuir a um indivíduo códigos de diferentes níveis, que por sua vez podem ser independentes ou inter-relacionados [2]. Na Figura 1 encontra-se a árvore da CIF com este exemplo evidente.

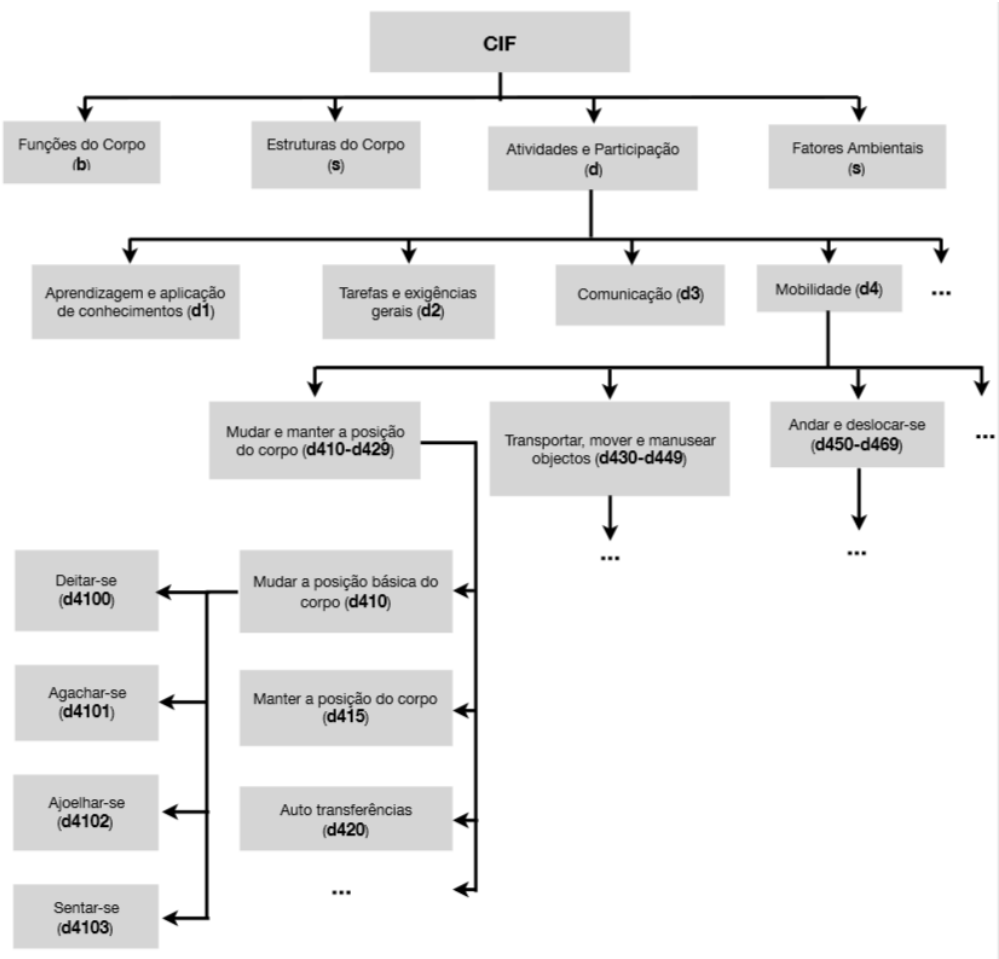

Figura 1 - Árvore da CIF [2].

As classificações das áreas da saúde são um aspeto fulcral e indispensável pois facilitam o levantamento, a consolidação, a análise e interpretação de dados, a formação de bases nacionais consistentes e a comparação de informações de saúde sobre a população ao longo do tempo, entre regiões e países. Existem diversos tipos de classificações, como por exemplo Classificação Internacional de Doenças e Problemas Relacionados à Saúde[3] (CID) para diagnósticos, a International Classification of Impairments, Disabilities, and Handicaps [4] (ICIDH) para procedimentos, Systematized Nomenclature of Medicine Clinical Terms[5] (SNOMED CT) para terminologias, entre outras. Diferenciando-se destas classificações encontra-se a CIF, onde uma alteração funcional poderá possuir diversos códigos de cada parte da classificação, ao contrário da CID, que é uma classificação de doenças na qual, em geral, cada diagnóstico possui apenas um código [4]. Enquanto a CID proporciona códigos para a mortalidade e morbidade, a CIF oferece códigos para retratar a variação completa de estados funcionais [5].

Do ponto de vista epidemiológico, a CIF representa um instrumento mais apropriado do que a CID-10, dado que possibilita que se conheçam os determinantes da incapacidade, fatores ambientais e fatores sociais. Orienta ainda os profissionais que trabalham com o processo de reabilitação na determinação de condutas terapêuticas. A CIF e o seu modelo são ferramentas clínicas e epidemiológicas importantes dado que direcionam o trabalho na prática da Fisioterapia e de outras especialidades envolvidas nos processos de reabilitação. A CIF fornece assim aos profissionais da saúde uma listagem completa dos processos de saúde, considerando um perfil funcional específico para cada indivíduo, dispondo de recursos para a construção de um tratamento centrado nas necessidades de cada um. A linguagem comum e uniformizada para aplicação universal padroniza conceitos e terminologias, facilitando assim a comunicação entre *stakeholders*, nomeadamente profissionais, investigadores ou pessoas com incapacidade [3].

No entanto, trata-se de um processo subjetivo uma vez que, a mesma função pode ser avaliada e classificada de forma diferente por cada grupo de profissionais, de cada clínica, hospital, região ou país. Isto torna difícil a recolha de dados estruturados para obter informações epidemiológicas sobre a funcionalidade, incapacidade e qualidade de vida das populações. O referido processo de tradução entre as informações recolhidas e os códigos fornecidos pela CIF referido denomina-se, no âmbito desta dissertação, codificação, que é uma tradução do conceito inglês *linking process.* O procedimento referido é habitualmente realizado por dois profissionais independentes que seguem determinadas regras, e onde divergências são resolvidas através da discussão chegando assim a um consenso, envolvendo por vezes um terceiro profissional. Agravando consideravelmente esta realidade, o grau de entendimento entre os profissionais está diretamente dependente do conhecimento que cada um possui sobre a CIF, da experiência, da educação ou treino profissional e do tipo de informação que pretendem traduzir. Isto significa que diferentes objetivos possam ser registados de formas distintas, dado que é um processo subjetivo e dependente de muitos fatores [6]. Por exemplo, o termo "identificar" pode estar relacionado com diversas situações dependendo do contexto. Pode ser identificar a mãe (código

[3] http://www.who.int/classifications/icd/en/
[4] http://www.who.int/classifications/ichi/en/
[5] http://www.snomed.org/

d1600) ou identificar imagens ou números (código d1370). No entanto, estes exemplos não se encontram na CIF, sendo responsabilidade dos profissionais avaliar e catalogar estes termos. Este processo como é feito manualmente e não é registado numa plataforma partilhada, apenas estes avaliadores terão acesso a esta codificação.

1.2 Objetivo

Com todos estes pontos mencionados anteriormente, é possível depreender que a codificação da CIF é uma tarefa bastante complexa, suscetível a erros e que exige um tempo excessivo por parte dos avaliadores, dado que não existe qualquer tipo de ferramenta de suporte informático, tendo sido realizada até ao momento de forma manual. Numa época em que praticamente tudo se encontra informatizado, onde cada vez mais mergulhamos numa sociedade digital, a tecnologia tornou-se indissociável da forma como as gerações atuais lidam e interagem com o mundo, sendo praticamente obrigatório ferramentas de suporte informático para este tipo de tarefas [6]. As transformações tecnológicas associadas ao computador e ao mundo virtual consolidam-se cada vez mais como parte indispensável do quotidiano nesta segunda década do século XXI. O facto da codificação da CIF ser feita manualmente até ao momento, teve impacto na evolução da mesma, uma vez que apenas a partir da prática e utilização se consegue detetar erros e constatar melhorias [7]. Por esta razão, sistemas tecnológicos tornam-se extremamente importantes para incentivar a utilização desta classificação, para facilitar o seu processo de codificação com ganhos na validação e fiabilidade [6].

Uma vez que os sistemas tecnológicos existentes não agilizam o processo de codificação, nem incluem uma interação direta com os profissionais, surgiu a necessidade do desenvolvimento de uma aplicação *web*, que preencha esta lacuna que existe a nível informático. Com este projeto pretende-se assim, dotar os profissionais de saúde com uma aplicação *web* que, com base em regras predefinidas, os auxiliem a codificar e registar as intervenções clínicas e os respetivos resultados de uma forma mais eficiente e menos propensa a erros de subjetividade. O sistema além de disponibilizar a consulta da CIF na língua portuguesa com uma pesquisa personalizada, deve possibilitar a submissão de propostas para instâncias da classificação.

Através desta plataforma, espera-se facilitar o processo de codificação e, consequentemente, reduzir o tempo despendido por parte dos avaliadores, aumentando ainda a adesão desta classificação e possibilitar a contínua evolução da CIF, disponibilizada até aos dias de hoje.

1.3 Estrutura da Dissertação

A dissertação encontra-se organizada em 6 capítulos. O primeiro capítulo contém uma introdução ao projeto, expondo a motivação e os objetivos. O segundo capítulo apresenta a história e importância da classificação CIF. O terceiro capítulo explica o estado de arte atual, onde são comparados os sistemas já existentes, realçando as suas vantagens e o que os diferencia. Ainda neste capítulo são analisadas as diferentes tecnologias existentes para o desenvolvimento de uma aplicação *web* e o motivo que levou à escolha das tecnologias utilizadas nesta solução. No quarto capítulo é revelado todo o processo de desenvolvimento da aplicação *web*. Inclui a definição dos requisitos funcionais e não funcionais, *user stories*, a arquitetura e a implementação, com a descrição de ficheiros e as suas funções principais. No quinto capítulo é demonstrado o protótipo da aplicação final, com imagens das páginas mais relevantes e as suas funcionalidades. Por fim, o último capítulo conclui a dissertação com os pontos chave do trabalho desenvolvido, evidenciando as principais contribuições e relevando um possível trabalho futuro.

CAPÍTULO 2

Funcionalidade e Incapacidade Humana

Este capítulo tem como objetivo instruir o leitor sobre a história e importância da classificação retratada neste projeto, a CIF. Inicia-se com a explicação de algumas definições e conceitos, e a forma como foram evoluindo ao longo do tempo, uma vez que são importantes na relação entre o ser humano com a medicina e a sociedade. Com o surgimento de diferentes pensamentos emergiram três modelos ao longo dos tempos, que influenciaram a definição destes conceitos, e deste modo é apresentado um resumo destes três. Por fim, são descritas as classificações que tiveram por base um destes modelos, a ICIDH e a CIF.

2.1 Funcionalidade e Incapacidade Humana

Em primeiro é necessária uma compreensão dos conceitos de funcionalidade e incapacidade humana na medida que, estes foram e são especialmente importantes na evolução da relação entre o ser humano com a medicina e a sociedade. A definição e a determinação da extensão do termo incapacidade tem vindo a ganhar uma grande notoriedade, desde que ocorreu um aumento da esperança média de vida, das doenças crónicas e das suas consequências [5]. A ausência de uma explicação concreta deste termo, é considerada um obstáculo para o progresso da saúde de indivíduos com deficiência. A necessidade de obter conhecimento acerca do que acontece com os pacientes após o diagnóstico, principalmente no caso de pessoas com doenças crónicas ou que tiveram acidentes que as deixaram incapacitadas, é muito relevante em termos de prestação de cuidados de saúde. A necessidade deste olhar mais amplo sobre a saúde humana levou ao surgimento do termo “funcionalidade humana”, que explica a inter-relação entre as estruturas e as funções do corpos, a atividade individual e a participação na sociedade [1].

O conceito funcionalidade humana é complexo pois representa todas as atividades que o ser humano é capaz de realizar, que permite a cada pessoa ser quem é, um corpo único, biológico e social capaz de perceber e atuar no mundo que a rodeia. Tudo aquilo que cada indivíduo consegue fazer como ler, aprender, trabalhar, conviver, andar e por aí em diante, ou seja, tudo aquilo que se possa imaginar perpetuar. No entanto, algumas pessoas possuem mais capacidades em relação a outras. Por exemplo, podem aprender mais rápido, dispor de uma melhor capacidade de conversação, ou de uma locomoção mais veloz. Ou seja, todo o ser humano é diferente e daí a complexidade de caracterizar a funcionalidade humana. Este último termo depende da articulação de vários conhecimentos, implicando a compreensão do indivíduo num ambiente que possui as suas múltiplas dimensões, onde a atividade humana é afetada pela interação com as propriedades dos vários ambientes e portanto, esse conhecimento apenas é possível num contexto de transversalidade e interdisciplinaridade [8].

Existe uma grande dificuldade em associar formalmente uma terminologia relativa à funcionalidade e à incapacidade, particularmente devido à ambiguidade conceptual existente ainda dentro deste campo. Diversos fundamentos ao longo do tempo têm afetado a definição destes dois conceitos, os quais se podem definir em três modelos: i) modelo médico; ii) modelo social; iii) modelo biopsicossocial [9].

2.2 Modelo Médico

A génese do modelo médico, surgiu na década de 60 e teve como berço a medicina. Retrata o interesse na doença, deficiência ou “anormalidade corporal”, e como estes fornecem algum nível de limitação funcional ou incapacidade. O corpo, de acordo com este ponto de vista, é visto um objeto de interesse científico, de classificação e intervenção e assim que é classificado como incapacitado ou dependente, o indivíduo é considerado uma vítima, precisando de cuidados adicionais, ajuda e atenção de terceiros. Como tal, o tratamento encontra-se no cuidado de pessoas experientes, ficando ao encargo de equipas multiprofissionais delinear as necessidades do indivíduo e as intervenções precisas para reduzir o impacto negativo de tal incapacidade [10]. Neste paradigma, a lesão, a doença ou a limitação física são considerados como a primeira causa de desigualdade social e dos malefícios com os quais as pessoas com deficiências se deparam [11].

Este modelo encontrava-se em vigor aquando do decorrer da guerra do Vietnam, o qual gerou uma grande revolta nos Estados Unidos da América, pois muitos indivíduos regressaram da guerra mutilados e incapacitados, e para os quais a medicina não tinha respostas completamente adequadas, sentindo-se a indignação por parte da sociedade uma vez que após defenderem o seu país, encontravam-se desamparados e sem as devidas ajudas e adequações médicas aos seus estados de saúde. A partir daqui, ocorreu um novo pensamento: a sociedade também era responsável por criar formas de integrar este tipo de pessoas. Criaram-se assim movimentos de mobilização popular e intelectual em torno de questões sociais, tais como os direitos humanos, racismo, pobreza, exclusão entre outros, abrindo assim as portas para outro tipo de modelo, o modelo social [6].

2.3 Modelo Social

O modelo social de incapacidade foca-se na deficiência como um problema concebido pela sociedade e não como uma característica do indivíduo [9]. A ideia principal está na estrutura social, onde a deficiência é vista como uma questão de vida em sociedade em detrimento de uma adversidade individual, transferindo assim a responsabilidade das desvantagens das incapacidades corporais da pessoa para a incapacidade da sociedade em antecipar e adaptar-se à diversidade. Este entendimento mais aprofundado pode ajudar na inserção familiar, laboral e social da pessoa limitada e melhorar significativamente a sua qualidade de vida [11].

Em suma, neste modelo, a incapacidade é compreendida como uma consequência biológica do funcionamento incorreto do organismo e desta forma o papel do médico é recompor essa disfunção corporal. Por outro lado, este modelo sugere que a definição de deficiência e da incapacidade surge de circunstâncias sociais e culturais particulares. Com isto existem ainda outros autores que referem que a incapacidade não possui um significado universal, apontando assim que em alguns idiomas e culturas não existe o termo “incapacidade” e que as distinções sociais podem ser categorizadas de diferentes formas. Os debates relativamente às teorias da incapacidade pendem a dividir-se e a polarizar-se nas conceções médica e social. Essa polarização conceptual tem perturbado a análise relacional dos conceitos deficiência e incapacidade [10].

2.4 Modelo Biopsicossocial

Por fim, o terceiro modelo denominado modelo biopsicossocial, tem vindo a afirmar-se progressivamente ao longo dos anos. O seu propósito é tentar integrar ambos os modelos anteriores, o modelo médico e o modelo social de incapacidade. Fornece uma visão completa do ser humano e das suas condições de saúde que integra as dimensões física, psicológica e social, passando a dar relevância ao conceito funcionalidade em detrimento do rótulo da deficiência [12].

Consequentemente, o modelo biopsicossocial também coloca em foco a necessidade da evolução tanto do profissional como do paciente, no que diz respeito às capacidades relacionais que permitem o estabelecimento de um vínculo adequado e uma comunicação eficiente. Esta comunicação é indispensável para assegurar que os problemas e preocupações dos pacientes são compreendidos pelos profissionais e que informações importantes, recomendações e o próprio tratamento sejam percebidos, lembrados e aceites pelos pacientes [12].

O ponto fulcral neste modelo não é apenas a doença em si e o seu tratamento, mas sim todas as noções que estão diretamente relacionadas com o adoecimento, sejam elas psicológicas, sociais, ambientais, fisiológicas, entre outras, as quais também necessitam

de ser consideradas em prol de um tratamento mais eficiente. Adicionalmente, este modelo fornece ainda informações essenciais com o intuito de apaziguar os receios dos pacientes quando são submetidos a procedimentos médicos. Essas informações podem ser relativas às consequências que poderão advir destes procedimentos, as melhores formas de reduzir estas, entre outras que possam ser relevantes para minimizar o *stress* do paciente. Relativamente ao tratamento, possibilita ao doente a escolha do melhor método, na sua opinião, a ser utilizado e tal escolha pode ser considerada pelos profissionais, proporcionando assim outra forma de apaziguar as ansiedades e depressões consequentes. Deste ponto de vista, é essencial a presença da dimensão subjetiva do doente, envolvendo os aspetos existentes entre a psicologia social e da saúde [12].

Existem diversos modelos que são utilizados ao nível da reabilitação e das áreas relacionadas e que advêm do modelo biopsicossocial, nomeadamente: i) modelo de incapacidade de Nagi; ii) a *International Classification of Impairments, Disabilities and Handicaps* (ICIDH); iii) o Processo de Produção de Desvantagem (*Disability Creation Process* (DCP)) que é uma aplicação do Modelo de Desenvolvimento Humano (*Human Development Model* (HDM)); e iv) a Classificação Internacional de Funcionalidade, Incapacidade e Saúde (CIF) [9]. Destes cinco modelos apenas o modelo da CIF e o seu antecessor (ICIDH) serão aprofundados nesta dissertação [9].

2.5 ICIDH

A evolução da medicina, no que diz respeito ao conhecimento e tecnologia médica, ajudou na eliminação ou no controlo de muitas doenças agudas e no aumento da esperança média de vida, o que levou consequentemente a um aumento expressivo da população idosa. Com isto foi dada uma maior relevância às doenças crónicas no seio da sociedade. Visando responder às necessidades de conhecer mais sobre as consequências das doenças e o seu impacto na vida diária das pessoas, a OMS publicou em 1980, a *International Classification Impairments, Disabilities, and Handicaps* (ICIDH), que traduzida para português: Classificação Internacional de Deficiências, Incapacidades e Desvantagens (CIDID) [9].

Esta identifica 3 dimensões: i) deficiência (*impairment*) - dimensão orgânica; ii) incapacidade (*disability*) - dimensão pessoal; iii) desvantagem (*handicap*) - dimensão social [9]. A Tabela 2 apresenta a definição de cada dimensão de acordo com o modelo ICIDH de 1980.

Tabela 2 - Definição das dimensões da ICIDH [6]

Deficiência	Qualquer perda ou anomalia psicológica ou anatómica. Entre as quais deficiências intelectual, psicológica, da linguagem e da audição.
Incapacidade	Redução ou incapacidade para exercer uma atividade considerada normal. Esta implica a incapacidade no comportamento, na comunicação, do corpo, da destreza, entre outros.
Desvantagem	Condição negativa que resulta de uma deficiência ou falta de capacidade, que limita ou impede o exercício considerado normal para os humanos, tendo em conta características individuais como a idade, o sexo e os fatores socioculturais. Pode incluir desvantagens de orientação, na independência física, na mobilidade, integração social, entre outras.

De acordo com estas descrições, uma deficiência pode refletir-se em uma ou mais incapacidades para efetuar determinados exercícios e/ou num conjunto de desvantagens para uma pessoa. Tendo em conta que um indivíduo com uma deficiência pode não possuir a autonomia para efetuar determinadas atividades quotidianas, a utilização de ajudas técnicas pode fazer a diferença, isto é, a aplicação de instrumentos específicos para funções de compensação (compensação ou amplificação de funções inexistente ou comprometidas) ou funções de substituição (instrumentos e estratégias alternativas). Posto isto, ajudas técnicas representam instrumentos que possibilitam a recuperação de funções corporais ou incrementam a autonomia de um indivíduo [6].

A ICIDH definia, como sequência linear, as condições decorrentes de uma doença: Doença → Deficiência → Incapacidade → Desvantagem [11]; e que estes conceitos encontravam-se relacionados, contudo eram independentes. Ou seja, algumas deficiências podem ser uma incapacidade ou não, e podem ou não se tornar numa desvantagem. Posto isto, não se pode olhar para todas as deficiências e afirmar que estas se traduzem automaticamente numa desvantagem ou numa incapacidade [6].

O modelo gerou bastantes críticas e polémicas principalmente devido ao conceito de desvantagem, levando assim a um processo de revisão promovido pela própria Organização Mundial da Saúde, que culminou na publicação da CIF [13] .

2.6 CIF

2.6.1 História

Ao longo dos anos, ocorreram diversas discussões entre estudos devido ao facto de conceitos distintos e termos chave, tais como incapacidade, limitação e função terem significados diversos e por vezes sobreposto, dificultando assim a comunicação, a discussão clínica e a investigação. O conceito funcionalidade em si ainda é alvo de grande variabilidade conceptual, dado que o termo pode ser aplicado a diferentes significados, assim como pode ser expresso utilizando terminologias distintas. Os termos *functionality*, *functioning*, *functional capability* por exemplo, são diferentes e no entanto todos se referem ao conceito funcionalidade [9].

Com isto, é natural que tenham sido elaborados na literatura ao longo dos anos, diversos comentários tanto positivos como negativos relativamente aos vários modelos da incapacidade. Tendo como base estes princípios, a OMS realizou uma grande revisão à ICIDH e em 2001 aprovou a CIF. Este modelo visa oferecer uma visão biopsicossocial coerente sobre a saúde e os estados da saúde [9].

2.6.2 Diferenças

Uma das principais diferenças entre a ICIDH e a CIF está na introdução de um novo modelo por parte da CIF, que privilegia a funcionalidade como componente da saúde e considera o ambiente como um facilitador ou um obstáculo para o desempenho das funções e tarefas de um indivíduo. A CIF introduz um paradigma que consiste numa abordagem mais ampla, biopsicossocial [3] e onde o termo saúde está associado a diversas áreas da vida, como a funcionalidade em si, o bem-estar e a qualidade de vida [9]. O aspeto primordial é o facto de conceptualizar a incapacidade como resultado de um conjunto de situações e condições que incluem o ambiente, as condições de vida bem como as pessoais. Assim, este modelo não vê o indivíduo como uma pessoa com uma deficiência, mesmo sendo provisória, dando destaque a todos os componentes que auxiliam e complicam a realização das suas funções, tanto biológicas como sociais. Isto demonstra uma mudança na visão baseada na doença, para uma visão baseada na funcionalidade da pessoa como componente essencial da saúde [3].

É destacado neste novo modelo a importância do meio ambiente. Enquanto que no ICIDH no meio ambiente existiam algumas ajudas técnicas que não faziam parte da funcionalidade, como já foi referido anteriormente, onde estas representavam instrumentos que possibilitam a recuperação de funções corporais ou incrementam a autonomia de um indivíduo (por exemplo uma cadeira de rodas para a mobilidade), na CIF tudo isso já faz parte da funcionalidade humana, assim como o meio ambiente. A funcionalidade está diretamente dependente do meio que rodeia o indivíduo e isto é efetivamente uma grande mudança relativamente ao ICIDH [3].

No modelo mais recente, o conceito “funcionalidade” é principalmente estudado no seu aspeto mais positivo, no entanto esta classificação permite também que se avaliem os graus de perda funcional. Tendo em conta todos os componentes da CIF, como as funções dos órgãos e sistemas, as condições ambientais, as limitações de atividade e de participação e as estruturas do corpo, a CIF consegue identificar o que um individuo pode ou não estar apto a fazer na sua vida diária [3].

2.6.3 Conceito

A CIF pertence à “A Família de Classificações Internacionais” da OMS (*WHO Family of International Classifications* (WHO-FIC)) que tem o propósito de incentivar a seleção adequada de classificações mundiais em diversas áreas da saúde. Estas auxiliam no levantamento, consolidação, análise e interpretação de dados; a criação de base de dados nacionais consistentes, e concedem a comparação de informações acerca de populações ao longo do tempo entre regiões e países [1].

Até ao momento tem havido uma tentativa de recolher, formatar e propor diversos modelos conceptuais a partir da interpretação e explicação dos conceitos incapacidade e funcionalidade, cujo maior exemplo encontra-se na dialética: modelo médico versus modelo social da incapacidade. É neste exemplo que a CIF se destaca e diferencia, promovendo uma perspetiva mais ampla, integrativa e universal dos conceitos, onde o indivíduo interage com o meio físico, social e biológico, em detrimento de uma visão reducionista de ambos os modelos, médico e social [14].

De acordo com a CIF, a funcionalidade e a incapacidade estão associadas às condições de saúde de uma pessoa, indicando o que ela “pode ou não efetuar na sua vida quotidiana”, tendo em conta as estruturas do corpo, as funções dos órgãos ou sistemas, assim como as limitações de atividades [1].

2.6.4 Terminologia e Conceitos

As terminologias e conceitos utilizados na CIF encontram-se dispostos na Tabela 3.

Tabela 3 - Conceitos chave da CIF. Adaptação: [1]

	Parte 1: Funcionalidade e Incapacidade			**Parte 2: Fatores Contextuais**		
Componentes	**Estruturas e Funções do Corpo**	**Atividade**	**Paticipação**	**Fatores Ambientais**	**Fatores Pessoais**	**Condição de Saúde**
Definição	Funções do corpo são as funções fisiológicas dos sistemas do corpo, incluindo as funções mentais. Estruturas do corpo são as partes anatômicas do corpo.	Realização de uma tarefa ou ação por um indivíduo.	Envolvimento numa situação da vida social.	Influências externas sobre a funcionalidade e a incapacidade	Influências internas sobre a funcionalidade e a incapacidade	Qualifica doenças, distúrbios, lesões ou traumas e inclui também circunstâncias como a predisposição genética, envelhecimento, *stress* ou anomalia congénita.
Aspeto Positivo	Integridade Funcional e Estrutural.	Atividade	Participação	Facilitadores	Não Aplicável	Não Aplicável
	Funcionalidade					
Aspeto Negativo	Deficiência	Limitação da atividade	Restrição da participação	Barreiras/ Obstáculos	Não Aplicável	Não Aplicável
	Incapacidade					

A CIF é uma classificação hierárquica que se baseia no corpo, no indivíduo e na sociedade. Subdivide-se em duas partes: i) Funcionalidade e Incapacidade; e ii) Fatores Contextuais. Cada uma destas partes subdivide-se ainda em dois componentes: i) a primeira incluí as funções e estruturas do corpo; e ii) a segunda em fatores ambientais e pessoais [9].

A descrição de cada subdivisão apresenta-se da seguinte forma:

Estruturas e funções do corpo

Estes conceitos estão relacionados, respetivamente, com as estruturas anatómicas e as funções fisiológicas. No modelo abordado, a incapacidade é definida como qualquer problema nas estruturas ou funções do corpo. Consoante certas condições, se existir alguma ajuda técnica que possa compensar determinada dificuldade ou após uma alteração das estruturas ou funções do corpo, pode não se verificar uma incapacidade para realizar certas atividades. Um exemplo disso é o caso de uma pessoa amputada, que com uma prótese, é capaz de locomover-se [9].

Atividades

Estão relacionadas com o conjunto de tarefas executadas pelo indivíduo. As dificuldades nas atividades são decretadas como limitações. Normalmente as limitações são causadas pelas alterações das funções do corpo, mas também podem ser consequência de barreiras ambientais. Quando a capacidade de uma pessoa para executar atividades é avaliada, as repercussões das alterações nas funções do corpo ou das barreiras ambientais tornam-se óbvias [9].

Participação

Corresponde à interação do indivíduo nas atividades do quotidiano e na sociedade. Quando existem dificuldades na participação, estas são rotuladas como restrições na participação. Uma restrição ocorre quando a pessoa não é capaz operar com aquilo que é considerado normal para o ser humano. Assim como as limitações nas atividades, as restrições na participação podem ser causadas por fraqueza, doença ou *handicap*, assim como por barreiras ambientais [9].

Fatores contextuais

Corresponde aos fatores ambientais e pessoais que podem auxiliar ou restringir a funcionalidade de uma pessoa. Relativamente aos fatores ambientais, estes correspondem ao mundo físico ou social, e os fatores pessoais correspondem aos elementos que caracterizam cada pessoa como um ser único e individual, como por exemplo a raça, o género, a educação, os estilos de vida, características pessoais e experiência de vida [9].

Os estados correspondentes à saúde de uma pessoa podem ser registados a partir da seleção de códigos de uma categoria adequados em conjunto com os qualificadores. Cada componente pode ser qualificado como positivo ou negativo, desta forma são contruídos códigos numéricos que caracterizam o grau de funcionalidade ou incapacidade, ou a importância que um fator ambiental apresenta (facilitador ou barreira) [9].

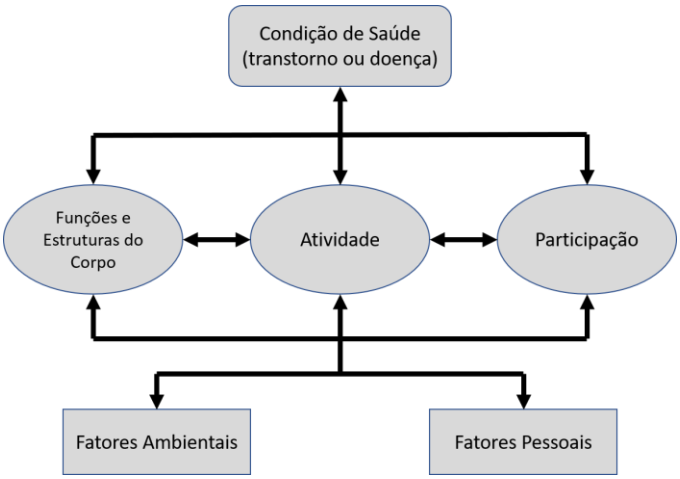

Figura 2 - Interação entre os componentes da CIF. Adaptação: [1]

Na Figura 2 é possível observar o espetro de interações e relações relatadas na CIF. A funcionalidade de uma pessoa é o fruto da relação complexa entre a condição de saúde e os fatores contextuais (ambientais e pessoais). Em vez de classificar as pessoas, a CIF interpreta as suas características, particularmente, as estruturas e funções do corpo, incluindo as psicológicas, as interações das pessoas com o meio que as rodeia (atividades e participação) e as características do meio ambiente (fatores contextuais), o que vai conceder a descrição do estado funcional do indivíduo, resultando assim na desvalorização dos aspetos negativos. Desta forma, este modelo deixa de classificar o ser como uma pessoa com deficiência, mesmo que seja momentânea, priorizando todos os componentes que auxiliam ou dificultam a execução das funções, tanto sociais como biológicas [9].

Consoante a CIF os componentes de funcionalidade e incapacidade podem apontar problemas (e.g. limitação na atividade, incapacidade ou restrição na participação) ou aspetos neutros/positivos afiliados ao conceito funcionalidade [9].

2.6.5 Aplicação

A expressão “canivete suíço” tem sido utilizada para retratar a CIF, visto que esta possui um vasto leque de ferramentas que possibilitam diversas abordagens. A CIF pode ser utilizada em vários ramos como a saúde, medicina do trabalho, previdência social, políticas públicas e estatísticas. A sua relevância pode ser imposta para as práticas clínicas, de investigação e do ensino. A nível clínico, dispõem-se a servir de modelo de atendimento multidisciplinar, auxiliando as várias equipas e os seus recursos, tais como médicos, psicólogos, terapeutas, assistentes socias, entre outros [1].

Uma das vantagens assinaladas para a aplicação do modelo é a possibilidade de uniformização de conceitos, resultando assim na utilização de uma linguagem padronizada facilitando a comunicação entre gestores, investigadores, profissionais de saúde, organizações da sociedade civil e utilizadores em geral [1].

Existe uma grande dificuldade na utilização adequada e completa da CIF devido à sua enorme variedade de recursos. Para facilitar a sua aplicação têm sido produzidos instrumentos que sintetizam a classificação. A partir da CIF, a OMS apresentou uma lista genérica que inclui apenas as condições consideradas mais relevantes a serem levantadas durante a avaliação de um paciente. Esta *checklist* é constituída por 152 categorias que representam os domínios mais importantes: i) 20 códigos das estruturas de corpo; ii) 57 da atividade e participação; iii) 38 das funções do corpo; iv) 37 dos fatores ambientais [1].

Para além deste instrumento, a OMS e o ramo de pesquisa da CIF criaram os “*Core Sets*”, que são um conjunto de categorias que descrevem a funcionalidade de um indivíduo com uma determinada condição de saúde. Estes podem ser resumidos ou abrangentes, dependendo da forma de como são utilizados. As versões resumidas são aplicadas apenas por um profissional de saúde, enquanto que as versões abrangentes são adotadas por equipas multiprofissionais. O objetivo dos core sets da CIF é resumir a avaliação de 1454 aspetos da funcionalidade das pessoas para apenas avaliar as categorias relevantes numa

determinada condição de saúde. Assim, para avaliar cada paciente foram escolhidas apenas 55 a 168 categorias, de acordo com a sua condição. Até ao momento, ainda é um instrumento em desenvolvimento e apenas serve para avaliar “o que” deve ser medido nos indivíduos com determinadas condições de saúde, mas não determinam “como” esses aspetos devem ser qualificados [15].

A área da Medicina Física e Reabilitação tem sido a mais explorada para a aplicação da CIF, mais especificamente na parte do acompanhamento do estado de saúde dos pacientes em tratamento. Quando há necessidade de associar dados de saúde ocupacional com os dados de previdência social, de acordo com alguns autores a CIF deve ser utilizada como uma base de informações padronizadas em estudos relativos à incapacidade profissional. Desta forma a CIF tem despertado maior interesse e atenção na sociedade relativamente às deficiências que certas pessoas possuem, fornecendo as bases para as políticas e disciplinas da Saúde Pública em relação aos indivíduos com deficiências. A CIF torna-se uma ferramenta bastante útil dado que permite responder a certas questões relevantes comparando o estado de saúde das pessoas que possuem deficiências e das que não possuem [2].

2.6.6 Classificação

Em termos de codificação, a CIF possui quatro domínios: i) funções e estruturas do corpo; ii) atividade; iii) participação; e iv) fatores ambientais. Cada um destes está organizado por capítulos. Este modelo contém 1424 categorias organizadas de acordo com um sistema alfanumérico. Cada uma das categorias começa com uma letra, a qual corresponde ao domínio do componente [16]:

- *Body Functions* (b) - Funções do corpo
- *Body Structure* (s) - Estrutura do corpo
- *Activity & Participation* (d) - Atividade e Participação
- *Environmental Factores* (e) - Fatores Ambientais

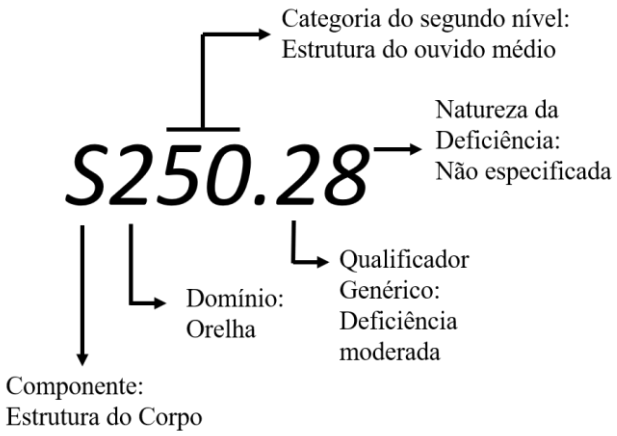

Figura 3 - Exemplo da estrutura de um código CIF. Adaptação: [16].

A Figura 3 exemplifica o formato geral de um código da CIF. Considerando o código do exemplo, s250.28, que regista uma pessoa com um problema moderado na estrutura do ouvido médio, o código pode ser dividido em diversos elementos onde [16]:

- 's' denota o componente, que neste caso é a estrutura do corpo.
- O primeiro dígito (2) denota o domínio, que no exemplo é a orelha e as estruturas relacionadas.
- O segundo e o terceiro dígito (50) representam a categoria de segundo nível, sendo a estrutura do ouvido médio.
- O primeiro dígito após a vírgula (2) corresponde ao qualificador genérico, indicando, neste caso, uma deficiência moderada no ouvido médio. Os qualificadores indicam o nível de um determinado parâmetro, como o nível de facilitadores ou barreiras, causado pela condição de saúde.
- O segundo dígito após a vírgula (8) indica, neste exemplo, que a natureza da deficiência não está especificada.

Dependendo das necessidades, pode ser utilizado mais que um código em cada nível para retratar uma situação de um paciente. Os códigos podem ser independentes ou inter-relacionados. Dentro da CIF, a condição de saúde de um indivíduo por ser representada por uma gama de códigos a partir dos domínios dos componentes da classificação. Cada aplicação tem disponível no máximo 34 códigos ao nível do capítulo (8 códigos de estruturas do corpo, 8 de funções do corpo, 9 de capacidade e 9 de desempenho) e 362 no segundo nível. Para o terceiro e quarto níveis, há até 1424 códigos disponíveis constituindo assim a versão completa da classificação [2].

2.6.7 Qualificadores

Segundo a OMS, o sistema de codificação da CIF apenas está completo após a utilização de um qualificador. Estes indicam informações sobre o estado da funcionalidade (a magnitude, localização e natureza de qualquer problema) e colocam-se após o código CIF, separado por uma vírgula decimal ou um sinal de +, "fechando" assim o código (xxx.00) [17].

Um código pode conter até 3 qualificadores, dependendo do componente a ser codificado. O primeiro qualificador é comum a todos os componentes e especifica a extensão do problema numa escala numérica que varia de 0 a 4 (Tabela 4). Pode ser uma deficiência de uma função do corpo, uma limitação nas atividades, uma restrição de participação ou até mesmo a inexistência de um problema de funcionalidade. Para além disto, ainda podem ser utilizados os dígitos 8 ("não especificado") e 9 ("não aplicável"), sendo o primeiro utilizado quando não existem informações suficientes sobre a categoria (i. e. O indivíduo tem um problema de visão, mas não se sabe se esse problema é ligeiro ou grave). O dígito 9 refere-se à utilização inapropriada da categoria (i.e. b650 funções da menstruação para um homem) [17].

Tabela 4 - Sistema de qualificação da CIF. Adaptação: [2].

Qualificador	Tadução	Exemplo de tradução semântica
xxx.0	NÃO há problema	(nenhum, ausente, insignificante, ...)
xxx.1	Problema LEVE	(leve, pequeno, ...)
xxx.2	Problema MODERADO	(médio, regular, ...)
xxx.3	Problema GRAVE	(grande, extremo, ...)
xxx.4	Problema COMPLETO	(total, ...)
xxx.8	não especificado	
xxx.9	não aplicável	

No caso dos componentes das Estruturas do Corpo para além do qualificador comum a todos os componentes, podem ser também utilizados dois qualificadores opcionais, um que especifica a natureza da deficiência e outro revela a localização, como lado esquerdo ou direito [17].

Quanto ao componente Atividades e Participação podem ser utilizados dois qualificadores: desempenho e capacidade. O primeiro explica o que um indivíduo realiza no seu ambiente habitual, enquanto que o segundo descreve a capacidade de uma pessoa para executar uma tarefa, onde o efeito do contexto está ausente ou é relevante. O qualificador de Capacidade presume a avaliação de uma pessoa sem qualquer assistência pessoal ou de instrumentos [17]. Tudo isto é possível verificar na Tabela 5.

Tabela 5 - Qualificadores da CIF. Adaptação: [2].

Componentes	Primeiro Qualificador	Segundo Qualificador
Body Functions **(b)** **Funções do corpo**	**Qualificador genérico** com a escala negativa usado para indicar a extenção ou magnitude da deficiência.	Nenhum
Body Structure (s) **Estrutura do corpo**	**Qualificador genérico** com a escala negativa usado para indicar a extenção ou magnitude da deficiência.	Utilizado para apontar a natureza da alteração na estrutura do corpo em questão: 0 - nenhuma mudança na estrutura 1 - ausência total 2 - ausência parcial 3 - parte suplementar 4 - dimensões anormais 5 - descontinuidade 6 - desvio de posição 7 - mudanças qualitativas na estrutura, incluindo retenção de líquidos 8 - não especificada 9 - não aplicável
Activity & Participation **(d)** **Atividade e Participação**	**Desempenho** Qualificador genérico. Problema no ambiente habitual do indivíduo.	**Capacidade** Qualificador genérico. Limitação sem assistência.
Environmental Factores **(e)** **Fatores Ambientais**	**Qualificador genérico** com uma escala negativa usado para indicar, respetivamente, a extenção das barreiras e facilitadores.	Nenhum

2.6.8 Vantagens

A utilização da CIF tem sido aplicada cada vez mais em diferentes domínios devido aos seus benefícios. A avaliação da funcionalidade, recorrendo a este modelo, fornece uma abordagem multidisciplinar, completa e centrada no indivíduo. Denota-se como uma ferramenta epidemiológica e clínica importante, especialmente na área da medicina de reabilitação, auxiliando a comunicação entre equipas multidisciplinares, para organizar o processo de reabilitação, determinar e avaliar objetivos, registos e documentação [9], dado que é uma linguagem comum e padronizada para aplicação universal, que uniformiza conceitos e terminologias, facilitando assim a comunicação entre profissionais, nomeadamente investigadores, pessoas com incapacidades ou, decisores políticos [18].

A CIF serve como modelo conceptual e teórico para a funcionalidade e incapacidade, tornando-a numa ferramenta essencial e que possibilita a explicação e descrição dos diversos fatores que influenciam o estado funcional dos indivíduos. Por outro lado, a CIF inaugurou um novo paradigma em que a visão negativa que anteriormente estava associada à deficiência, agora é trocada por uma visão neutra e onde a funcionalidade é interpretada como o resultado da interação do indivíduo com o meio ambiente onde se inclui. Com o avanço da tecnologia, que se encontra incluída no domínio de fatores ambientais, a CIF tem uma grande importância para perceber qual o impacto da tecnologia na funcionalidade de qualquer pessoa [9].

Por fim, representa um marco conceptual para descrever a saúde e os estados relacionados com a mesma, constituindo um valioso instrumento de utilidade prática na Saúde Pública [14].

2.6.9 Desvantagens

A Classificação Internacional de Funcionalidade, Incapacidade e Saúde permite representar o perfil de funcionalidade do ser humano, pois oferece um ordenamento sistemático e significativo de todas as informações relativas à funcionalidade. No entanto este processo requer a tradução dessas informações recolhidas nas categorias da CIF. Para tal existem dois métodos: i) utilizando ferramentas ou instrumentos e qualificadores adequados para traduzir as informações reunidas; ou ii) codificando a observação clínica diretamente nas categorias e qualificadores da CIF. Estas informações recolhidas, tanto de uma forma como da outra, podem ser traduzidas nas categorias da CIF para descrever a situação funcional do indivíduo [17].

A combinação destes dois métodos ou de apenas um dá origem ao perfil de funcionalidade. Para a escolha do método e do nível de detalhe (número de categorias) a serem utilizados devem ser considerados o âmbito e a relação custo-benefício. O tempo preciso para recolher as informações para a codificação do perfil não depende da CIF, mas sim da experiência profissional do indivíduo, do seu conhecimento e da complexidade das ferramentas de avaliação utilizadas. Implica aspetos como a granularidade (quantidade de

códigos imprescindíveis para o perfil), a experiência do profissional e o alinhamento direto do instrumento de avaliação com o modelo CIF. Quanto à utilização de ferramentas de avaliação, a tradução de códigos é mais simples com aquelas que são baseadas na CIF, como a WHODAS 2.0[6], do que com aquelas que não são. Para além do método utilizado, os recursos humanos dependem do contexto clínico onde o perfil é efetuado, pois com equipas multidisciplinares, a divisão da codificação entre os vários profissionais diminui significativamente o tempo e facilita a carga de trabalho, no entanto aumenta a desconcordância na avaliação [17].

Para completar um perfil de funcionalidade que seja representativo de todos os domínios da saúde, todos os componentes da CIF devem ser tidos em conta. Os fatores que necessitam de ser considerados são a escolha entre um perfil homogéneo ou um perfil que dê enfase a áreas específicas, escolher se vai limitar o número de categorias a serem utilizadas ou não e por fim que códigos serão utilizados. Cada opção aplicada terá diferentes pontos fortes e fracos e não existe nenhum método que seja mais adequado para todas as situações. Cada pessoa deve escolher a solução que melhor se ajusta à sua finalidade e configuração. Isto proporciona uma maior especificidade, mas pode ser mais difícil ou complexo de gerir, necessitando ao mesmo tempo de um maior conhecimento sobre a codificação da CIF [17].

O referido processo de tradução entre as informações recolhidas e os códigos fornecidos pela CIF referido denomina-se, no âmbito desta dissertação, que é uma tradução do conceito inglês *linking process.* Um exemplo deste procedimento encontra-se representado na Tabela 6.

Tabela 6 - Exemplo do linking process [6].

Objetivos	Ações/Estratégias
Explorar um brinquedo com as 2 mãos, durante 5 minutos, sem o levar à boca, com orientação do adulto. **d1310 - aprender através de ações simples com um único objeto.** **[d1310, e310]**	Explorar brinquedo grandes, de bater, carregar, arrastar. **e410; b7602; d445 - manipulação/uso da mão.** **[d1201; e11520]**
	Posicioná-lo sentado de lado, seja no colo, banco ou chão para brincar. **e410; b760** **[d4153; d9200; e1150; e310]**
	Colocar os objetos no lado direito ou esquerdo alternando para facilitar as transferências de cargo. **e410 + b7158 (carga + estabilidade, co-contração); b760** **[d4106, e11520; e310]**
	Modelar diferentes possibilidades de exploração do mesmo brinquedo, de forma a que a criança repita ações como: - bater; - carregar; - arrastar. **e410; b140; d130 - variabilidade da exploração** **[d130, e410]**

[6] http://www.who.int/classifications/icf/whodasii/en/

Este procedimento é realizado normalmente por dois profissionais independentes que seguem determinadas regras, e onde discordâncias são resolvidas através da discussão chegando assim a um consenso, e por vezes pode até envolver um terceiro profissional. Para agravar mais a situação, o grau de entendimento entre os profissionais está diretamente dependente do conhecimento que cada um possui sobre a CIF, da experiência, da educação ou treino profissional e do tipo de informação que pretendem traduzir [19]. Isto implica que diferentes objetivos possam ser registados de formas distintas, dado que é um processo subjetivo e dependente de muitos fatores.

A codificação do exemplo na Tabela 6 foi realizado por duas terapeutas do Núcleo Regional Norte da Associação Portuguesa de Paralisia Cerebral e revisto (códigos a vermelho) por duas investigadoras da Universidade de Aveiro. Com este é exemplo é possível verificar a subjetividade e discordância que pode ocorrer num processo de codificação da CIF.

CAPÍTULO 3

Estado de Arte

Este capítulo tem como objetivo explicar os estudos e os tópicos relevantes que foram necessários para a realização deste projeto. Em primeiro lugar são apresentadas plataformas com funções semelhantes às pretendidas e realizada uma comparação entre estas, destacando as diferenças que potenciam o objetivo deste projeto. De seguida, é efetuado um estudo comparativo entre algumas tecnologias que existem para a execução de uma aplicação *web* e o porquê da escolha efetuada. Posteriormente foi realizado um levantamento do conteúdo teórico sobre as tecnologias abordadas e como cada uma se apresenta na pilha de desenvolvimento. De seguida foi descrito o funcionamento da MEAN[7] *Stack* no desenvolvimento *web*, juntamente com suas vantagens e desvantagens. Por fim, são relevados quais os *softwares* complementares escolhidos para o desenvolvimento desta aplicação e quais os seus benefícios.

3.1 Trabalhos semelhantes

Os tópicos desta secção apresentam as plataformas mais conhecidas relativas à CIF (Rehadat-ICF[8], ICF Browser[9] e ICF Update Plataform[10]). Foi efetuada uma análise e comparação sobre as mesmas, destacando os problemas e diferenças existentes, que potenciam o objetivo deste projeto. Com este estudo é possível averiguar as falhas e as necessidades que existem nesta área. Foram avaliadas três aplicações *web* que se encontram descritas a seguir.

[7] http://mean.io/
[8] https://www.rehadat.de/de/
[9] http://apps.who.int/classifications/icfbrowser/Default.aspx
[10] https://bit.ly/2OWGmHI

3.1.1 Rehadat-ICF

Esta plataforma, denominada Rehadat-ICF tem como objetivo relacionar o sistema de informação Rehadat sobre os tópicos de deficiência, trabalho e reabilitação com o guia da CIF. Rehadat é um sistema de informação para a reabilitação profissional (ajudas técnicas, investigação, legislação e serviços), em que cada código do último nível da CIF apresenta um conjunto de resultados provenientes de uma pesquisa realizada sobre o sistema Rehadat utilizando a CIF. Os resultados são constituídos pelos seguintes campos [20]:

- Literatura – Podem ser artigos, livros ou publicações;
- Produtos de apoio – Descrições e números de produtos de apoio, imagens e endereços de referência;
- Casos de estudo.

Possibilita um campo de pesquisa sobre a classificação, no entanto, sem qualquer capacidade de aplicar filtros para realizar uma pesquisa mais personalizada. Além disto, dispõe de um conjunto de artigos, revisões e relatórios de investigação sobre a CIF no setor da literatura. Na secção da pesquisa, proporciona uma seleção de projetos e cientistas de reabilitação sobre a implementação e desenvolvimento da CIF.

É um sistema que não contribui para a comunicação entre especialistas nem possibilita a evolução da CIF, visto que este não é o seu objetivo. A única funcionalidade que apresenta alguma semelhança com a Plataforma de Codificação da Classificação Internacional da Funcionalidade, Incapacidade e Saúde (PCCIF) é o facto de permitir a pesquisa e consulta sobre a classificação CIF.

3.1.2 ICF Browser

A OMS disponibilizou no seu *website* um navegador que permite a consulta da classificação CIF em cinco linguagens diferentes assim como um método de pesquisa. A pesquisa pode ser filtrada por título, descrição, inclusões e exclusões. No entanto foram encontrados alguns problemas e incoerências:

- Na seleção da linguagem/versão disponibiliza uma opção “ICF – English” e outra “ICF 2017 – English”, não sendo explícito qual a versão da primeira opção.

- Quando é efetuada uma pesquisa e não são encontrados resultados o sistema bloqueia e não permite repetir a pesquisa ou retomar à página anterior, sendo necessário selecionar um domínio da CIF na árvore que se encontra do lado esquerdo da página, para tornar a reassumir a pesquisa e a consulta.

- A pesquisa não inclui o filtro para apenas procurar códigos, e quando se introduz o código b110, é apresentada a mensagem “*no results found from your search*”.

Em geral a plataforma dispõe de poucas funcionalidades e algumas falhas que a tornam de certa forma dispensável. O único ponto semelhante à PCCIF é a consulta sobre a CIF e uma pesquisa, com menos filtros e com problemas funcionais.

3.1.3 ICF Update Plataform

ICF Update Plataform é um sistema que permite aos utilizadores introduzirem e reverem propostas para as atualizações da CIF. É aberto para qualquer pessoa, no entanto diferentes utilizadores possuem níveis de autorização distintos. Para além de recolher propostas, possibilita a navegação pela árvore da CIF e verificar o que foi proposto em cada código. A plataforma gere todo o processo de revisão desde o momento em que a proposta é inserida pela primeira vez no sistema, até ao momento da sua remoção. Para poder usufruir da plataforma é necessário estar registado [21].

Após o *login* é apresentado um menu com as seguintes opções: *Home, ICF, Search/Filter/Report, All Groups, User Profile e Documents.* A página inicial está dividida em duas partes: do lado direito está uma seção que é utilizada para um acesso rápido às propostas, enquanto que do lado esquerdo estão as notícias relacionadas com a plataforma. A página ICF permite o acesso à classificação CIF e a submissão das propostas através de um botão "*add*" disposto ao lado de cada código.

A terceira página do menu está dividida também em duas partes. Do lado esquerdo é possível observar várias pesquisas predefinidas. Por exemplo, o link "*Accepted Proposals*", fornece automaticamente uma lista de propostas aceites, mas que ainda não estão implementadas. Do lado direito da página, encontram-se opções para pesquisar e filtrar propostas. Para além da funcionalidade de pesquisa de texto simples, o sistema permite filtrar os resultados de acordo com diversos parâmetros diferentes que podem ser combinados.

Na página "*All Groups*" estão dispostos todos os grupos fechados e as suas propostas. Se pertencer a um dos grupos aparece uma opção adicional no menu principal com o nome do respetivo grupo. Dentro desta é possível constatar os outros membros do grupo e discutir sobre propostas realizadas.

"*User Profile*" como o nome indica é onde estão as informações do utilizador que por sua vez podem ser editadas. Por fim, a página Documentos disponibiliza o acesso ao guia de utilização da plataforma.

Este sistema possui um objetivo diferente do da PCCIF, uma vez que o seu propósito é que os utilizadores enviem propostas para a atualização da árvore da CIF, de forma a que esta possa crescer, abranger mais conceitos detalhados, e consequentemente mais códigos. No entanto, não tem impacto no auxílio da catalogação em intervenções clínicas e na redução do tempo despendido no processo codificação por parte dos avaliadores. Também existem mais plataformas deste género, lançadas pela OMS, para outras classificações como a CID-11 e CID-10. Porém a ICF Update Plataform é aquela que se encontra mais "pobre" em termos de funcionalidades e aparência.

3.2 Estudo das tecnologias

O desenvolvimento de aplicações *web* é uma combinação do desenvolvimento do *front-end* e do *back-end*. Também conhecido como lado do cliente, o *front-end* abrange a criação de uma *interface* gráfica do utilizador para que os clientes (utilizadores) possam interagir com a aplicação. Envolve a utilização de ferramentas e tecnologias como o Hypertext Markup Language[11] (HTML), Cascading Style Sheets[12] (CSS) e JavaScript[13]. O HTML é uma linguagem de marcação que oferece a estrutura para uma página *web*, definindo como esta deve ser apresentada ao utilizador. O CSS, é uma linguagem que fornece estilo e melhorias visuais aos documentos escritos em HTML. JavaScript é a linguagem mais avançada destas três e realiza a manipulação do HTML Document Object Model (DOM), por forma a assegurar uma *interface* dinâmica aos utilizadores. Proporciona a criação de mensagens *pop-up*, validação de entradas de formulários e alteração do *layout* com base em eventos (*mouse clicks* ou texto escrito). Todas estas tecnologias são controladas por um navegador que oferece a *interface web* do *front-end*. O *back-end,* também conhecido como o lado do servidor, abrange a criação de Application Programming *Interface*s (APIs) e bases de dados para servir o cliente. As tecnologias geralmente consistem em linguagens como Hypertext Preprocessor[14] (PHP), Ruby[15], Python[16], Java[17], Node.js[18] e diferentes frameworks que fornecem auxílio [22].

Uma aplicação *web* é assim constituída por todo um conjunto de programas que implementam um sistema de informação segundo um paradigma Cliente/Servidor. Para ocorrer a comunicação entre o Cliente e o Servidor é necessário a utilização de protocolos de comunicação. Estes variam consoante as diferentes tarefas e camadas. Alguns exemplos de protocolos comuns utilizados nas aplicações *web* são o Hypertext Transfer Protocol[19] (HTTP), o Transmission Control Protocol / Internet Protocol[20] *(*TCP/IP), File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP) [22].

Construir um bom sistema de gestão de base de dados para armazenar informação é uma parte crucial no desenvolvimento de aplicações *web*. Uma vez que este permite que os utilizadores criem, guardem, atualizem e consultem dados da aplicação [22].

Posto tudo isto, o desenvolvimento da aplicação *web* implica um estudo prévio sobre quais as linguagens e tecnologias a serem utilizadas, atendendo aos objetivos e expectativas do que foram propostos neste projeto. A sua construção deve ter em conta a robustez, segurança, eficiência, ser *user friendly*, entre outros pontos [23].

[11] https://www.w3.org/html/
[12] https://www.w3.org/Style/CSS
[13] https://www.javascript.com/
[14] http://php.net/
[15] http://www.ruby-lang.org/pt/
[16] https://www.python.org/
[17] https://www.java.com/pt_BR/
[18] https://nodejs.org/en/
[19] https://www.w3.org/Protocols/rfc2616/rfc2616.html
[20] http://www.tcpipguide.com/

3.2.1 Base de Dados

O primeiro aspeto estudado foi a base de dados, dado que é o principal apoio de toda a aplicação e que irá determinar quais as tecnologias a serem utilizadas. Existem diversos tipos de bases de dados, sendo que os dois maiores são as bases de dados relacionais e as bases de dados não relacionais ou Not Only Structured Query Language (NoSQL) [24].

Bases de Dados Relacionais

Estas bases de dados fornecem um formato rigidamente estruturado, baseado em tabelas relacionadas entre si. Uma tabela consiste em linhas e colunas nas quais a coluna possui uma entrada para uma categoria específica e as linhas contêm uma instância para os dados definidos de acordo com essa categoria. A linguagem padrão é o Structured Query Language (SQL). Duas das suas principais desvantagens são o custo e a escalabilidade [25].

A implementação de um sistema de Sistema de Gestão de Banco de Dados (SGBD) pode ser caro, moroso de configurar e manter, principalmente em grandes organizações. Para configurar uma base de dados relacional, é necessário adquirir um *software* especial, altamente sofisticado que requer pessoas com uma formação adequada. Se a empresa for grande é requerido uma base de dados mais robusta e uma proteção contra o acesso não autorizado. Os custos resultantes de uma má conceptualização também são enormes, pois a construção de uma base de dados defeituosa acarreta consequências desastrosas numa organização [26].

Atingir escalabilidade e elasticidade é um enorme desafio para este tipo de bases de dados, pois foram projetadas num período em que os dados a serem mantidos eram pequenos, organizados e ordenados. Assim como as taxas de transação cresceram ao longo da última década, também os volumes de dados a serem armazenados aumentaram massivamente. As bases de dados relacionais foram projetadas para executarem num único servidor para preservar a integridade dos mapeamentos das tabelas e evitar problemas de computação distribuída. Se um sistema necessitar de ser redimensionado, este *design* "obriga" os clientes a comprarem um *hardware* maior, mais complexo e mais caro com maior poder de processamento, memória e armazenamento. Os *upgrades* também são um desafio, uma vez que a organização tem que passar por um longo processo de aquisição e, em seguida, colocar o sistema *offline* para realizar uma alteração [27].

Bases de Dados Não Relacionais

As bases de dados NoSQL surgiram como resultado do crescimento exponencial da Internet e do surgimento de aplicações *web*. Emergiram de empresas como a Amazon[21], a Google[22] e o Facebook[23], que lidam com uma quantidade massiva de dados. Este tipo de

[21] https://www.amazon.com/
[22] https://www.google.pt/
[23] https://www.facebook.com/

base de dados em vez de ser restrito pelos limites das arquiteturas de um único servidor, é projetado numa escala massiva em sistemas distribuídos. As bases de dados NoSQL podem ser escaladas horizontalmente, isto é, executadas em diversos servidores que trabalham em conjunto, cada um compartilhando parte da carga. Com esta abordagem, uma base de dados pode operar em centenas de servidores, num *hardware* barato e num ambiente qualquer. Outra vantagem encontra-se no momento em que ocorre a falha de um nó (servidor), os outros recebem a carga de trabalho, eliminando o único ponto de falha. Outra diferença encontrada neste tipo de bases de dados é a flexibilidade que oferece ao nível dos dados, uma vez que não necessita de obedecer a uma conceção prévia do seu formato [28]. Novas propriedades podem ser adicionadas a uma entidade do sistema a qualquer momento. Proporciona uma agilidade para criar e testar novas funções sobre novas informações [27]. Podem ser agrupadas em quatro formas distintas consoante a sua estratégia de armazenamento de dados [29]:

Chave-Valor – É utilizada uma *hash table* onde existe uma chave exclusiva e um ponteiro para um item específico de dados. Este é o modelo mais simples e fácil de implementar e não possui um esquema predefinido. No entanto é ineficiente quando apenas se pretende consultar ou atualizar um valor, entre outras desvantagens. Alguns exemplos são a Amazon SimpleBD[24] e o Oracle Berkeley DB[25].

Coluna – São utilizados para armazenar e processar quantidades de dados distribuídos muito grandes em diferentes máquinas. Os dados são organizados em células agrupadas logicamente em famílias de colunas. As colunas podem ser criadas em tempo de execução ou na definição do esquema, sendo que este último é predefinido. Alguns exemplos são a Cassandra[26] e HBase[27].

Orientado a grafos – Utilizado para lidar com conjuntos de dados estruturados, semiestruturados ou não estruturados muito grandes. É composto por três componentes: os nós, que são os vértices do grafo; os relacionamentos, que são as arestas que ligam os nós; e por fim as propriedades que são os atributos que representam características dos nós e relacionamentos. Ajuda as organizações a aceder, integrar e analisar dados de diversas fontes. Neste caso é possível apontar como exemplo o Neo4j[28] e o AllegroGraph[29].

Documentos – São bases de dados flexíveis no tipo de conteúdo, visto que não têm um esquema predefinido. Armazenam, retornam e gerem informação orientada a documentos, também conhecidos como dados semiestruturados. Os documentos são blocos de JavaScript Object Notation[30] (JSON) ou Extensible Markup Language[31] (XML), onde em vez de colunas com nomes e tipos de dados, contêm uma descrição do dado e o valor para essa descrição. Representam uma especialização da base de dados chave-valor, onde um documento é guardado e lido utilizando uma chave. Para além desta

[24] https://aws.amazon.com/pt/simpledb/
[25] https://bit.ly/2ERPE33
[26] http://cassandra.apache.org/
[27] https://hbase.apache.org/
[28] https://neo4j.com/
[29] https://franz.com/agraph/allegrograph/
[30] https://www.json.org/
[31] https://www.w3.org/XML/

funcionalidade, oferece diferentes formas de consultar documentos com base no seu conteúdo. Cada documento pode possuir a mesma ou uma estrutura completamente diferente e são agrupados em coleções. Algumas das suas vantagens inclui a modelação flexível dos dados, elevado desempenho no armazenamento e nas consultas. Alguns exemplos desta base de dados são o MongoDB[32] e o CouchDB[33].

Considerando os dados a serem utilizados neste projeto, foi selecionada uma base de dados não relacional. Uma vez que os dados possuem uma estrutura variável, o esquema da base de dados pode alterar ao longo da conceção da aplicação e os dados podem crescer exponencialmente ao longo da utilização da aplicação, este tipo de base de dados é o mais adequado visto que oferece todas estas possibilidades. Relativamente ao *software* foi escolhido o MongoDB, essencialmente devido ao facto de ter sido lecionado em algumas unidades curriculares possibilitando uma utilização mais eficiente, dado que já existe algum conhecimento prévio. Para além disto, a escolha também recaiu sobre os seus benefícios pois permite criar aplicações rapidamente, manipular diversos tipos de dados e gerir aplicações de forma mais eficiente em escala. Constatando ainda que o desenvolvimento é simplificado, uma vez que os documentos mapeiam naturalmente para linguagens de programação modernas e orientadas a objetos. A adoção massiva por parte de diversas empresas em todo o mundo também foi um fator decisivo na escolha [30].

3.2.2 Tecnologias *Front-End*

Para efetuar a escolha das *frameworks* a serem utilizadas no lado do cliente, é necessário inicialmente determinar qual a linguagem. Seria muito extenso falar sobre todas as linguagens que permitem o desenvolvimento de uma aplicação *web*, portanto apenas irá ser abordada a razão pela qual foi escolhida o JavaScript. Algumas das razões pela qual esta linguagem foi a eleita deve-se ao facto de ser a linguagem mais conhecida no desenvolvimento *front-end*, é a única que é executada de forma nativa no navegador e também pode ser utilizada no lado do servidor. Quase todo o *front-end* é uma combinação de JavaScript, HTML e CSS, onde o primeiro permite manipular os elementos das páginas, validar formulário e até recuperar dados num servidor com a ajuda do Asynchronous JavaScript and Extensible Markup Language[34] (AJAX) [31].

Existe um grande número de estruturas de desenvolvimento da *web* e bibliotecas baseadas em JavaScript, no entanto as tecnologias mais populares do lado do cliente são o Angular[35] e o React[36] [32]. A principal vantagem destas estruturas é que elas permitem criar *User Interfaces* (UIs) altamente intuitivas [33].

[32] https://www.mongodb.com/
[33] http://couchdb.apache.org/
[34] http://api.jquery.com/jquery.ajax/
[35] https://angular.io/
[36] https://reactjs.org/

Angular

Angular é uma plataforma que facilita o desenvolvimento do *front-end* de aplicações *web* em HTML e utiliza uma linguagem que compila JavaScript, o TypeScript[37]. É mantida pela Google e por uma comunidade de *developers*. Adota o padrão Model-View-Controller (MVC) para renderizar (transformar para algo visível) os componentes numa aplicação de página única. A versão inicial do AngularJS foi lançada em 2012 [34]. Algumas vantagens e desvantagens encontram-se descritas na Tabela 7.

Tabela 7 - Vantagens e desvantagens do Angular [35].

Vantagens	Desvantagens
• **TypeScript** O principal aspeto do Angular é a utilização de TypeScript, que é um superconjunto do JavaScript. É totalmente compilado para JavaScript e ajuda a identificar e eliminar erros comuns aquando a escrita do código, dado que fornece mecanismos avançados para auto completar código, navegação e refatoração.	• **Aprendizagem** Para programadores que nunca utilizaram TypeScript anteriormente é necessário aprender e gastar algum tempo apenas neste ponto. A arquitetura da *framework* também necessita de um estudo e uma compreensão mais aprofundada.
• **Performance** Os resultados de testes efetuados ao Angular apresentam um desempenho impecável e rapidez em termos de renderização, animação e acessibilidade do navegador a todos os componentes. Contém vários fatores que ajudam a tornar a aplicação mais rápida. O principal é a injeção de dependência hierárquica e suporte do Angular Universal, que é um serviço que permite renderizar a *view* da aplicação num servidor em vez de no *browser* do cliente.	• **Detalhado e complexo** Embora a arquitetura baseada em componentes seja um dos pontos fortes do Angular, a forma como são geridos é muito complexa. Por exemplo, é necessário cinco arquivos para um único componente, sendo fundamental injetar as dependências e declarar as interfaces do ciclo de vida do mesmo. Concluindo, é despendido imenso tempo de desenvolvimento a realizar operações repetitivas.
• **Angular CLI** A interface da linha de comandos do Angular é uma ferramenta que possibilita a geração rápida de código, adicionando componentes, testando-os e implementado os mesmos diretamente a partir desta.	• **DOM regular** O Angular manipula DOM diretamente, o que o torna mais lento e menos eficiente comparativamente ao React.
• **Ecossistema poderoso** Como esta *framework* já existe a algum tempo, está repleta de pacotes, *plug-ins*, complementos e ferramentas de desenvolvimento. Além disto, como é uma *framework* bastante utilizado é possível encontrar todo o tipo de tutoriais e suporte para problemas ou erros encontrados no desenvolvimento.	• **Documentação do Angular CLI** Apesar do Angular CLI ser uma linha de comandos bastante útil, existe pouca informação na sua documentação oficial.

[37] https://www.typescriptlang.org/

React

É uma biblioteca JavaScript para desenvolver aplicações *web*. É mantida pelo Facebook e também por uma comunidade de *developers*. Adota um padrão MVC para renderizar os componentes numa aplicação de página única. Passou a ser *open source* na JSConf[38] em 2013 [36]. A Tabela 8 apresenta algumas vantagens e desvantagens desta tecnologia.

Tabela 8 - Vantagens e Desvantagens do React. [37], [38]*:*

Vantagens	Desvantagens
• **Separação total dos dados e da apresentação** O React fornece pouco mais do que uma camada de apresentação. Embora os componentes da biblioteca tenham um conceito de "estado", esta é a melhor utilização para um armazenamento efêmero.	• **Não é uma framework** Mesmo sendo uma vantagem também se apresenta como uma desvantagem, pois quando é necessário realizar algo rapidamente, pode ser frustrante.
• **Fácil aprendizagem e utilização** É fácil de aprender e fornece um conjunto de documentação e tutoriais úteis. Caso o programador possua um conhecimento prévio de JavaScript consegue aprender e começar a utilizar o React em apenas alguns dias.	• **Pouca documentação** Devido a atualizações frequentes é difícil manter uma documentação que as acompanhe. Para piorar, muitos contribuintes não consideram necessário escrever uma documentação adequada devido ao ritmo acelerado do desenvolvimento.
• **Não é uma framework** É uma biblioteca que fornece um método declarativo para definir componentes da interface do utilizador. O ReactDOM é uma biblioteca associada que fornece a renderização e DOM deffing. O Redux é outra biblioteca que proporciona um armazenamento de dados.	• **Elevado ritmo de desenvolvimento** Como foi referido no ponto anterior, com um ambiente em constante mudança, os developers necessitam de acompanhar todos os recursos novos, o que pode ser insuportável.
• **DOM virtual** Normalmente no desenvolvimento de aplicações que contém muita interação por parte do utilizador e atualização de dados, é necessário considerar cuidadosamente como a estrutura da aplicação irá ter impacto na performance. Com o DOM virtual, que como o nome indica é uma representação virtual do DOM, qualquer alteração na view é executada primeiro no DOM virtual. Esta abordagem oferece enorme flexibilidade e aumento no desempenho. O React também calcula quais as alterações necessárias no DOM, evitando assim operações dispendiosas, realizando atualizações de forma eficiente.	• **JSX** É uma extensão de sintaxe que permite misturar HTML com JavaScript e apresenta uma curva de aprendizagem ingreme, podendo ser frustrante.

Após este estudo comparativo entre as duas ferramentas para o desenvolvimento do *front-end* foi eleito o Angular. Ambos possuem as suas vantagens e desvantagens e são excelentes tecnologias, no entanto existia um maior interesse sobre a aprendizagem do

[38] https://jsconf.com/

Angular, mais especificamente a versão 2.0. Na Tabela 9 é possível observar algumas das características tanto do Angular como do React.

Tabela 9 - Características do Angular 2 e do React. Adaptação:[39].

	Angular 2	React
Linguagem	TypeScript	JSX, ES5, ES6
Conceção	JavaScript dentro do HTML	JavaScript Centric
JavaScript	Mais	Menos
Falhas	*Runtime*	*Compile-time*
DOM	Regular	Virtual
Binding	Bidirecional	Unidirecional
Templating	Ficheiros ts	Ficheiros jsx
MVC	Sim	*View* apenas
Renderização	Lado do servidor	Lado do servidor
Curva de Aprendizagem	Alta	Baixa
Abstração	Forte	Forte

3.2.3 Tecnologias *Back-End*

Durante este estudo foi possível apurar um conjunto de tecnologias que funciona muito bem como um todo, considerando as escolhas efetuadas até este ponto. Trata-se de uma *stack* (pilha) bastante conhecida e poderosa, denominada de MEAN *Stack*. MEAN é o acrónimo para MongoDB, ExpressJS[39], AngularJS e Node.js [40]. Foi projeta para oferecer aos *developers* uma forma organizada e rápida de desenvolver aplicações *web* robustas e sustentáveis, com módulos úteis como o Mongoose[40], *pre-bundled* e configurados. Tenta principalmente cuidar de pontos de conexão entre as estruturas existentes e resolver problemas de integração [41]. Os detalhes destas tecnologias encontram-se na secção 3.3.

Principais vantagens da solução MEAN *Stack* [42]:

- Oferece uma abordagem moderna e eficaz para o desenvolvimento *web*
- Utiliza o poder dos *Single Page Application* (SPA), não exigindo a atualização completa de uma página *web* a cada pedido ao servidor
- Todas as tecnologias utilizadas são gratuitas e *open-source*.
- Simplicidade e estrutura comum que proporciona.

[39] https://expressjs.com/
[40] https://mongoosejs.com/

    - MongoDB oferece mais flexibilidade, escalabilidade e a possibilidade de um crescimento exponencial de dados.
    - Node.js fornece uma melhor relação para executar o servidor.
    - Express ajuda a uniformizar a forma como a aplicação *web* é construída.
    - Angular oferece uma forma simples de adicionar funções interativas e componentes baseados em AJAX.
- Utilização de uma única linguagem: JavaScript.

Em suma, o recurso a estas tecnologias juntas produz resultados com elevado desempenho, escalabilidade, menos *overheads* (recursos necessários como CPU e memória) e oferecem simplicidade no desenvolvimento de aplicações *web*.

Para realizar o mapeamento dos objetos do MongoDB no servidor, foi escolhido o Mongoose, que é uma biblioteca do Node.js que proporciona uma solução baseada em esquemas para modelar os dados da aplicação. Dispõe de um sistema de conversão de tipos de dados, validação, criação de consultas, guardar e apagar dados. Proporciona um mapeamento de objetos semelhante ao Object-relational mapping (ORM) ou Object Document Mapper (ODM), ou seja, traduz os dados da base de dados para objetos JavaScript para que possam ser utilizados posteriormente. Define esquemas com os tipos dos dados e de seguida permite a criação de um modelo baseado nesse esquema. Esse modelo é depois mapeado para um documento MongoDB. Esta escolha deve-se ao facto de permitir evitar lidar com coleções dinâmicas sem estrutura definida, uma vez que possibilita a opção de trabalhar com esquemas definidos. Além disso, implementa a validação e outros recursos que garantem a consistência do mesmo [43].

3.3 MEAN *Stack*

Como foi referido na secção anterior (3.2.3), a solução escolhida para a implementação da aplicação *web* foi o conjunto de tecnologias baseadas em JavaScript denominado MEAN *Stack*. Desde o cliente, ao servidor e à base de dados, esta solução é *full* JavaScript.

Tabela 10 - Descrição das tecnologias utilizadas na MEAN [22].

Framework	Descrição
MongoDB	Sistema de Base de Dados
Express	*Back-end web framework*
Angular 2	*Front-end framework*
NodeJS	Ambiente de execução do *back-end*

O termo MEAN *Stack* foi utilizado pela primeira vez pelo engenheiro Valeri Karpov, em abril de 2013 numa publicação do *blog* oficial do MongoDB. A expressão foi adotada pela sua equipa para trabalhar numa maratona de programação utilizando MongoDB, ExpressJS, Angular JS e Node.js [44].

Um exemplo simples do funcionamento da MEAN *Stack* pode-se observar na Figura 4. O Cliente realiza uma operação, que irá despoletar um pedido HTTP no Angular direcionado ao Node.js, onde será analisado. O Express recebe os dados do pedido e efetua a devida operação sobre a base de dados MongoDB, retornando os dados ou uma mensagem para o Node.js. Este por sua vez vai retornar os dados numa resposta HTTP para o Angular, que irá disponibilizá-los ao Cliente [45].

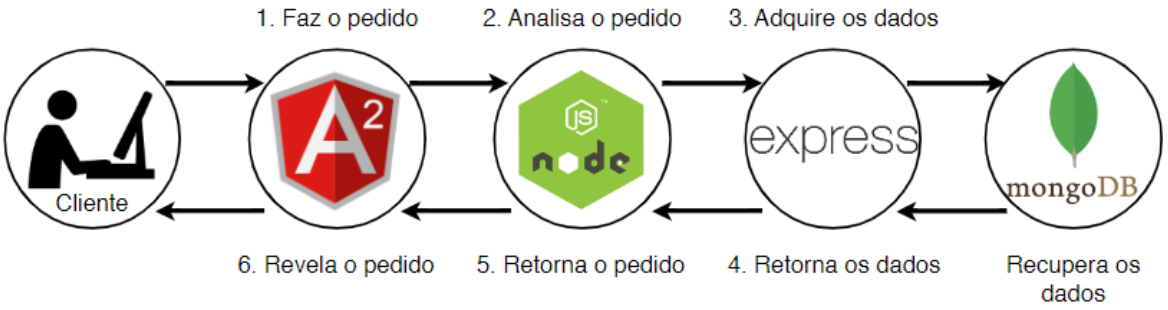

Figura 4 - Funcionamento da MEAN Stack. Adaptação: [46].

3.3.1 MongoDB

MongoDB é um *software* de bases de dados não relacionais, *open* source escrito na linguagem C++, que utiliza um modelo de dados orientado a documentos. Foi criado por Dwight Merriman e Eliot Horowitz, quando criavam aplicações *web* na DoubleClick, que é uma empresa de publicidade agora detida pela Google Inc. Ao longo do processo de desenvolvimento descobriram problemas de escalabilidade no desenvolvimento nas bases de dados relacionais. De acordo com Merriman, o nome da nova base de dados derivou da palavra *humongous*, que assenta na ideia de suportar uma grande quantidade de dados. A base de dados começou a ser desenvolvida em 2007 pela 10gen, atual MongoDB Inc. e a primeira versão foi lançada em 2009 [45].

A base de dados utiliza um armazenamento de documentos e troca de dados utilizando Binary JavaScript Object Notation[41] (BSON), que fornece uma representação binária de documentos baseados em JSON. A fragmentação automática permite que os dados de uma coleção sejam distribuídos por vários sistemas, para uma escalabilidade horizontal à medida que o volume de dados aumentam [45].

As estruturas dos documentos podem ser complexas dado que se situam entre os modelos de armazenamento de chaves – excecionalmente rápidos e escaláveis – e entre os tradicionais Sistema de Gestão de Banco de Dados Relacional (SGBDR) – que disponibilizam profundidade e amplos tipos de consultas. Para além deste suporte eficaz relativo às consultas, característico dos SGBDRs, a utilização de BSON capacita a indexação de qualquer tipo de dados, proporcionando um maior desempenho. As consultas

[41] http://bsonspec.org/

realizadas à base de dados são as mais utilizadas e quanto mais complexas forem, mais elevado é o custo de processamento. As consultas no modelo relacional necessitam de percorrer a tabela, coluna a coluna, quando não existe indexação dos dados. Relativamente ao MongoDB, é necessário percorrer toda a estrutura existente e, consequentemente, quanto maior for a estrutura, maior será o tempo de processamento despendido na mesma. A indexação de uma determinada informação gera um ficheiro. Estes ficheiros normalmente são mais pequenos que a estrutura em si e, portanto, necessitam de menos tempo para a sua leitura, dado que não é imprescindível que o processo leia todo o ficheiro de indexação, no caso de encontrar o que se pretende antes de o terminar. Outra justificação para a rapidez nas consultas no MongoDB, deve-se ao facto de este pré-alocar o espaço do ficheiro, prevenindo a fragmentação do mesmo e assegurando um armazenamento compacto e eficiente. Além disto, os ficheiros da base de dados são divididos em ficheiros mais pequenos de tamanhos pré-definidos. O primeiro com tamanho de 64 Megabytes (MBs), o segundo com 128MB e assim sucessivamente até 2 Gigabytes (GBs). Se forem necessários mais, o sistema continuará a utilizar ficheiros com tamanho de 2GB. Dado que não deixa ficheiros fragmentados nem com demasiada informação, este processo torna-se relevante para bases de dados volumosas, pois não perde desempenho ao fazer leituras em diversas zonas do disco [47].

3.3.1.1 Documentos

Um documento é a unidade básica de armazenamento de dados do MongoDB. Em vez de utilizar tabelas e linhas como uma base de dados relacional, o MongoDB é construído sobre uma arquitetura de coleções e documentos. Os documentos contêm conjuntos de pares chave-valor (campo: valor) [45]. São criados utilizando o JSON, no entanto no processo de armazenamento é utilizado BSON, como foi referido anteriormente, que oferece uma maior eficiência [48].

Cada documento dispõe de um identificador único (ID), que pode ser dado pelo utilizador quando o documento é criado, caso contrário é gerado automaticamente pela base de dados. O nome dos campos não pode conter pontos finais, "*null*" ou começar pelo símbolo "$". O nome do campo `_id` é reservado para a chave primária, o seu valor deve ser único na coleção e pode ser de qualquer tipo de dados menos *array* [48].

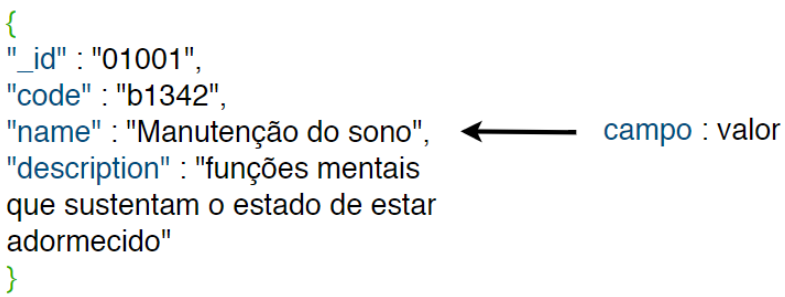

Figura 5 - Exemplo de um documento do MongoDB. Adaptação: [49].

3.3.1.2 Coleção

Uma Coleção é um grupo de documentos MongoDB (Figura 5). É equivalente a uma tabela das bases de dados relacionais [50]. Os documentos dentro de uma coleção podem ter campos diferentes devido ao facto do MongoDB ser uma base de dados *Schema-free*. Isto é, em bases de dados relacionais como o MySQL[42], o *schema* define a organização/estrutura dos dados na base de dados. Contrariando esta fórmula, o MongoDB não requere definir nenhuma estrutura. Isto possibilita armazenar documentos com diferentes campos na mesma coleção [51].

Os nomes das coleções devem começar por letras ou *underscore* e podem conter números. No entanto não podem exceder os cento e vinte e oito carateres [51].

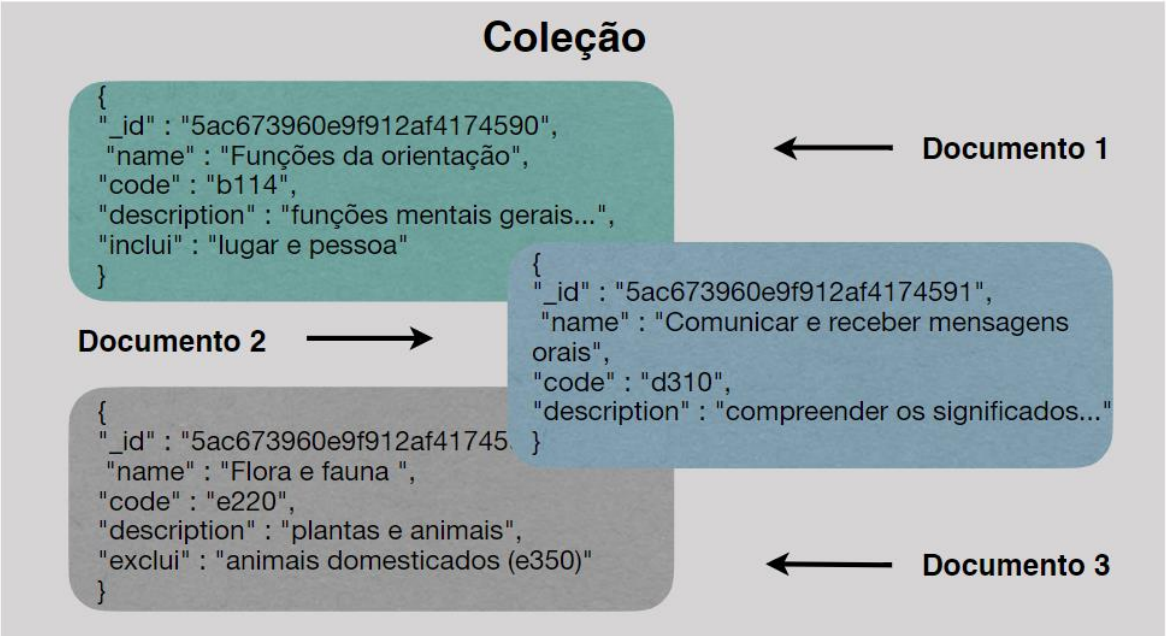

Figura 6 - Apresentação pictórica de Coleções e Documentos no MongoDB. Adaptação: [51].

3.3.1.3 Arquitetura

O MongoDB fornece escalabilidade através da implementação de *sharding* e replicação. Estas duas técnicas permitem ao MongoDB distribuir e duplicar os dados necessários [52].

Particionamento/Sharding

Sharding é um método para particionar os dados, que é utilizado quando a Coleção da base de dados encontra-se com um tamanho absurdo e, consequentemente, o

[42] https://www.mysql.com/

desempenho da aplicação está a degradar-se (Figura 6). Neste caso o MongoDB oferece este recurso nativo, que divide os dados da coleção entre vários servidores [53].

Existem dois métodos de abordar o crescimento do sistema: escalar verticalmente ou horizontalmente (Tabela 11).

Tabela 11 - Definição dos métodos do crescimento do sistema [53].

Método	Descrição
Escalabilidade Vertical	Aumentar a capacidade de um único servidor, através da adição de um CPU mais poderoso, aumentando a memória RAM, ou aumentando a quantidade/tipo do disco. Quando existem limitações na tecnologia disponível, uma única máquina pode não ser suficientemente poderosa para uma determinada carga de trabalho.
Escalabilidade Horizontal	Divisão do conjunto de dados do sistema entre vários servidores, e caso seja necessário, incluir servidores adicionais para aumentar a capacidade. Apesar da velocidade ou da capacidade global de uma única máquina possa ser baixa, cada máquina lida com um subconjunto de dados, oferecendo uma melhor eficiência quando comparada a um único servidor de alta velocidade e capacidade. O aumento da capacidade apenas requer a adição de servidores, conforme necessário, o que tende a ter um custo geral mais baixo.

Uma base de dados MongoDB distribuída consiste em três componentes principais [52]:

- *Shards*/Fragmentos – Referem-se aos servidores que guardam os subconjuntos de dados;

- *Query Routers* – São a *interface* entre a aplicação do cliente e os *shards*. São responsáveis pelo *routing* dos pedidos de leitura e escrita a *shards* apropriados. Para aceder e consultar o fragmento, a consulta utiliza informações fornecidas pelos *config servers*;

- *Config Servers* – Apenas guardam metadados sobre os *shards*, por exemplo, a localização dos dados e outras definições de configuração.

Estes componentes abordam como os dados são armazenados e recuperados, mas não como o processo de distribuição é realizado. A distribuição dos dados é realizada no nível da Coleção, fornecendo ao utilizador mais controlo dos dados. Assim, tanto as Coleções que são particionadas ou não, são suportadas pela mesma base de dados. A distribuição de dados é baseada nos *sharding keys* e na estratégia escolhida. Cada documento na Coleção fragmentada possui um campo que corresponde à *shard key*. Esta chave representa o critério utilizado para distribuir os documentos nos servidores e a estratégia de fragmentação refere-se ao algoritmo que determina qual o *shard* em que o documento irá ser colocado. Atualmente, existem duas estratégias disponíveis: *Range* ou *Hash*. De

acordo com o *Range*, a coleção será dividida em intervalos derivados da *shard key*. Por outro lado, o *Hash* é quando é MongoDB envia a *shard key* para uma função hash e o resultado desta determina a localização dos dados. Cada opção abrange o seu próprio conjunto de vantagens e desvantagens, e a escolha entre estas duas estratégias deve ser efetuada cuidadosamente [52].

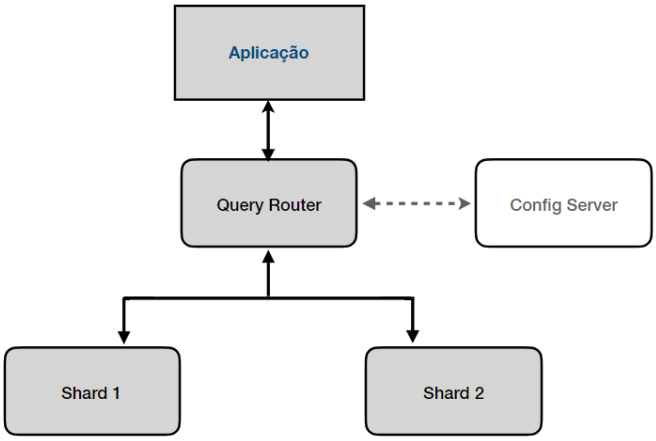

Figura 7 - Exemplo de uma aplicação distribuída. Adaptação: [52].

Replicação

A replicação consiste em manter cópias idênticas dos dados em vários nós do *cluster*, de modo a assegurar a disponibilidade dos dados de forma automática, no caso de algum nó falhar (Figura 8). É baseado numa configuração mestre/escravo dado que dentro de um conjunto de réplicas, os servidores portam papeis atribuídos [52]:

- Nó primário (mestre) – é o destinatário de todos os pedidos de escrita e armazena as alterações no seu *log* (registo dos eventos de replicação) de operações.
- Nós secundários (escravos) – Utilizam o *log* do nó primário para replicar as alterações para o seu próprio conjunto de dados, garantindo a consistência.

Num esquema mestre/escravo, se o nó primário desliga, o sistema possui apenas um único ponto de falha. O MongoDB lida com este problema através de um protocolo de comunicação denominado *heartbeat*, onde os membros do conjunto da replicação enviam *pings* ou *heartbeats* entre eles a cada dois segundos. Caso um dos membros não responda, dentro de 10 segundos é considerado morto. Através deste protocolo, os nós secundários são informados quando o mestre morre e realizam uma eleição para eleger um novo nó primário. Além destes dois tipos de nós ainda existem um terceiro, conhecido como nó arbitro, que é único no conjunto de réplicas e o seu propósito é apenas desempatar no caso de um empate nas eleições [52].

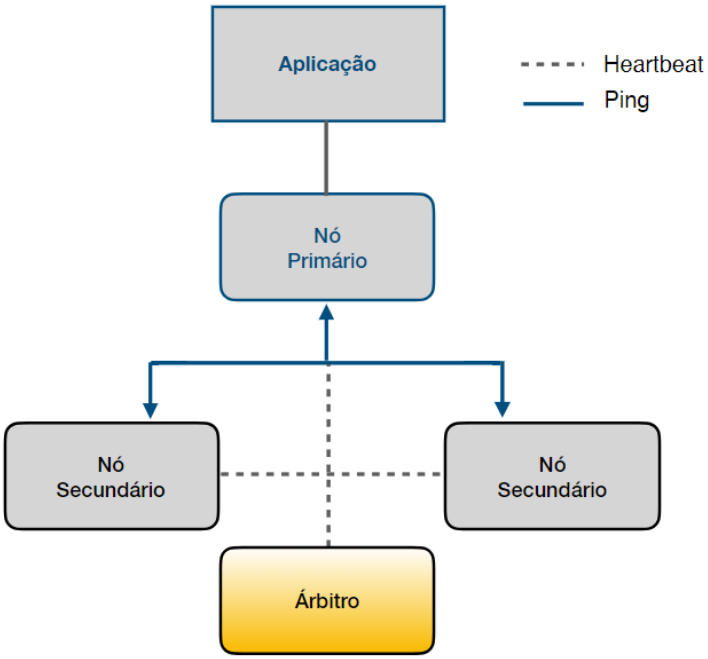

Figura 8 - Exemplo de um sistema replicado. Adaptação: [52]

3.3.1.4 Vantagens

Algumas das vantagens da utilização do MongoDB são:

- *Base de dados flexível devido à sua característica schema-less.*

MongoDB é uma base de dados de documentos na qual uma coleção contém diferentes documentos. O número de campos, conteúdo e tamanho do documento podem diferir [54].

- *Rapidez.*

É cem vezes mais rápida que uma base de dados relacional, proporcionando facilidade no acesso aos documentos por indexação [55].

- *Não dispõe da complexidade dos joins* [54].

- *Profunda habilidade de consultas*.

Suporta consultas dinâmicas nos documentos utilizando uma linguagem de consulta baseada em documentos que é quase tão poderosa como o SQL [54].

- *Escalabilidade.*

É um base de dados horizontalmente escalável. Quando o tamanho dos dados já é muito elevado, pode ser distribuído por diferentes máquinas [55].

- *Alta disponibilidade.*

Detém recursos como a replicação e Grid File System (GridFS) que ajudam a aumentar a disponibilidade dos dados e consequentemente o desempenho [55]. "*GridFS é uma*

especificação para armazenar e recuperar arquivos que excedem o limite de tamanho do documento BSON de 16 MB" [56].

- *Acesso rápido.*

Utiliza memória interna para armazenar o conjunto, permitindo um acesso mais rápido aos dados [54].

- *Suporte a consultas Ad-hoc.*

O MongoDB possui um recurso muito avançado para consultas ad-hoc (consultas com o propósito de obter informações conforme a necessidade, que dependem de uma variável [57]) [55].

- *Configuração do ambiente simples.*

É mais fácil configurar o MongoDB do que o SGBDRs e oferece ainda o JavaScript *Client* para consultas [55].

3.3.2 Node.js

Node.js é uma plataforma para construir aplicações *web,* que utiliza JavaScript como linguagem de *back-end,* ou seja, do lado do servidor [58]. Dado que esta é uma das linguagens mais utilizadas na programação de aplicações *web* do lado do cliente, torna a integração entre o cliente e o servidor mais simples [59]. É baseado no JavaScript Engine V8[43], que é um interpretador de JavaScript *open source* criado pela Google em C++ e que é utilizado pelo motor de busca Google Chrome[44]. Com o Node.js a linguagem JavaScript será utilizada não só pelo lado do *browser* (como normalmente ocorre), mas também pelo lado do servidor [60].

Foi apresentado em 2009 na JSConf por Ryan Dahl e é mantido pela fundação Node.js em parceria com a Linux Foundation. A inspiração veio quando Dahl estava a fazer *upload* de arquivos no Flickr[45] e ao rever a barra de progresso do *upload*, percebeu que o motor de busca necessitava de consultar o servidor *web* pois não sabia quanto do arquivo tinha sido carregado [60]. Node.js hoje em dia é utilizado por grandes empresas, nomeadamente LinkedIn[46], Netflix[47], Paypal[48] ou Nasa[49] [61].

O Node.js foca-se na *performance* e no baixo consumo de memória com o intuito de suportar processos do servidor de longa duração. Permite criar servidores rápidos e escaláveis, evitando a complexidade existente na dependência de *multithreading*, como

[43] https://v8.dev/
[44] https://www.google.com/chrome/
[45] https://www.flickr.com/
[46] https://pt.linkedin.com/
[47] https://www.netflix.com/
[48] https://www.paypal.com/
[49] https://www.nasa.gov/

muitos outros ambientes de desenvolvimento dependem deste para a execução de processos concorrentes na lógica do negócio do servidor. Esta plataforma opera apenas em *single-thread*, utilizando um modelo baseado em eventos e de I/O (input e output) não bloqueante (Figura 9) [59].

Utiliza operações de I/O não bloqueantes, assíncronas ou orientadas a eventos, registando uma função de *callback,* que é invocada quando uma tarefa é concluída prevenindo deste modo, que a aplicação não fique bloqueada. Tendo como exemplo uma função para ler um ficheiro, esta começa a ler e retorna imediatamente para o ambiente de execução para que a próxima instrução possa ser executada. Quando o arquivo está totalmente lido, vai ser chamada a função *callback* com o conteúdo do mesmo como parâmetro. Assim não existe nenhum bloqueio ou espera pela leitura do ficheiro. Isto torna o Node.js altamente escalável, pois consegue processar um alto número de pedidos sem necessitar de esperar que as funções retornem os resultados [59]. É uma opção que reduz imenso tempo e recursos para processar dados de diversas fontes. Quando existe uma carga muito grande de leitura e escrita de dados, este ambiente de execução consegue entregar um resultado bastante satisfatório com menos esforço [58].

Por fim, esta tecnologia integra um gestor de módulos, Node Package Manager[50] (NPM), que permite fazer a integração de módulos e pacotes externos que tenham sido desenvolvidos para qualquer programador partilhar ou reutilizar de forma simples. Isto possibilita desenvolver aplicações compostas por um conjunto de pequenos módulos, que são mantidos pelos seus autores [59].

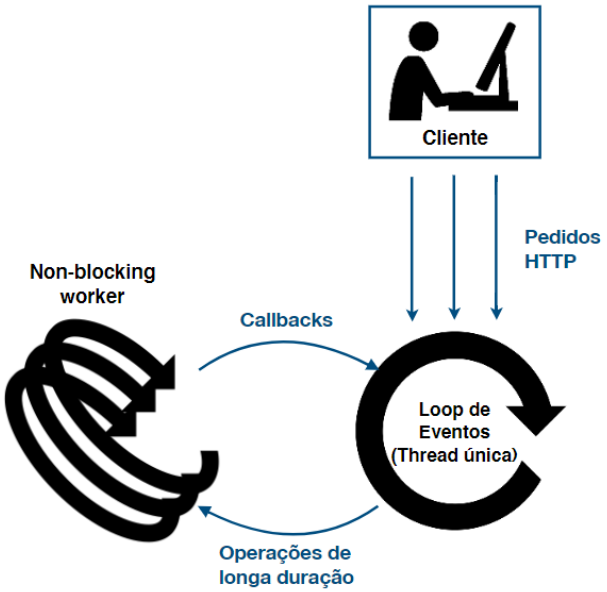

Figura 9 - Processamento de operações em Node.js. Adaptação: [59]

[50] https://www.npmjs.com/

3.3.2.1 Características

Assíncrono e orientado a eventos

Todas as APIs da biblioteca do Node.js são assíncronas/*non-blocking*. Por outras palavras, indica que um servidor baseado em Node.js nunca terá de esperar pela API para retornar os dados. Este passa para a próxima API e através de um mecanismo de notificação de eventos do Node.js, o servidor obtém a resposta do pedido da API anterior [58].

Única thread mas altamente escalável

Utiliza um modelo *single threaded* com *loop* de eventos. O mecanismo de eventos ajuda o servidor a responder de uma forma não bloqueante e torna o servidor altamente escalável, o que não acontece com os servidores tradicionais que criam *threads* limitadas para lidar com os pedidos. O Node.js utiliza um programa *single threaded,* tornando-o capaz de responder a milhares de pedidos simultâneos, sem os rejeitar ou criar *deadlock* (quando um ou mais processos ficam bloqueados indefinidamente, obrigando os outros a esperar que estes terminem as suas operações [62]). O mesmo não acontece com os servidores tradicionais, como por exemplo o Apache HTTP[51] [58].

Rápido

Dado que é construído sobre o interpretador V8 JavaScript, a biblioteca Node.js é muito rápida na execução do código [58].

Sem Buffering

As aplicações Node.js nunca guardam temporariamente os dados numa área reservada de memória, em vez disso, enviam os dados em partes [58].

Licença

Foi lançado sobre a licença MIT[52], que é uma licença de *software* grátis criada pelo Instituto de Tecnologias de Massachusetts [58].

3.3.2.2 Quando utilizar

Aplicações de mensagens instantâneas ou de chat

Uma das melhores situações para utilizar o Node.js encontra-se na criação de aplicações de *chats* ou ferramentas onde participem multiutilizadores, dado que existe um grande tráfego e utilização de dados em tempo real distribuídos por diferentes dispositivos. O Node.js consegue lidar muito bem com este tipo de cenários, visto que estes apresentam

[51] https://httpd.apache.org/
[52] http://www.mit.edu/

uma comunicação assíncrona e ele já suporta *WebSockets* através de bibliotecas como o Socket.IO[53] [63].

APIs escaláveis

Uma vez que uma das características do Node.js é utilizar operações I/O não bloqueantes, é uma ótima opção para criar APIs que sejam altamente escaláveis. Principalmente quando se pretende criar aplicações SPA é muito importante a questão da escalabilidade e do tempo de resposta das APIs, dado que existe uma vasta possibilidade de que o I/O de rede aumente significativamente, e por conseguinte, o servidor tenha que lidar com uma grande quantidade de pedidos simultâneos, sem perder *performance* ou consumir uma imensa quantidade de recursos (memória e processador) do servidor [63].

Streaming de dados

Os dados gerados em tempo real e com um fluxo contínuo são referidos como *Streaming* de dados. Quando é necessário lidar com estes, o Node.js é uma boa opção novamente devido à sua abordagem I/O não bloqueante. Este economiza imenso tempo e recursos para processar dados de fontes diferentes. Uma vez que existe uma carga muito grande de leitura e escrita de dados, o Node.js oferece um resultado satisfatório com pouco esforço, tanto em termos de codificação como de *hardware* necessário [63].

Internet das Coisas (IoT)

Segundo uma pesquisa efetuada pela fundação do Node.js em 2016, 96% dos indivíduos envolvidos no desenvolvimento de projetos Internet of Things (IoT) estão a utilizar JavaScript/Node [63].

Outros

Outras formas de utilizar esta plataforma é, por exemplo, no desenvolvimento de *dashboards* para a monitorização de aplicações ou em *back-end* de jogos [63].

3.3.3 Express

Express é uma *framework* flexível para aplicações *web* hospedada dentro da plataforma Node.js, escrita em JavaScript. Contém um conjunto robusto de características para desenvolver aplicações *web* e *mobile*, entre as quais um sistema MVC intuitivo, sistema de *routing* e um executável para a geração de aplicações [64]. Foi lançada em 2010 por TJ Holowaychuk e inspirada na *framework* Sinatra[54], no sentido de ser minimalista com diversas características disponíveis em *plugins*. Reduz consideravelmente as dificuldades da configuração de um servidor *web* para lidar com pedidos e respostas do cliente. Fornece um desenvolvimento rápido e simples de aplicações *web*, que não seria possível utilizando

[53] https://socket.io/
[54] http://sinatrarb.com/

apenas o Node.js [65]. Devido a esta simplicidade tem sido adotada por diversas grandes empresas como o Paypal[55], Uber[56] e Apiary[57] [22] [66].

Algumas das suas principais características [64]:

- Permite configurar o *middleware* para responder a pedidos HTTP.
- Define uma tabela de rotas que é utilizada para executar diferentes ações baseadas em HTTP e Uniform Resource Locator (URL).
- Permite renderizar dinamicamente as páginas HTML com base na transmissão de argumentos para os *templates.*
- Possibilita configurações comuns de aplicações *web*, como a porta a ser utilizada para a conexão, e a localização dos modelos utilizados para renderizar a reposta.

É aconselhável utilizar esta *framework* essencialmente nos casos em que as aplicações *web* possuem múltiplas funcionalidades para além de um serviço Representational State Transfer (REST). Por exemplo onde a lógica do negócio é complexa, quando há a necessidade da utilização de *middlewares* ou de renderização de *templates*. De outro modo, não é aconselhável utilizar o Express em serviços onde estas funcionalidades não sejam utilizadas e onde a implementação da lógica da aplicação requere uma grande repetição do código para a manipulação de recursos distintos [59]. Um servidor Express é constituído por três blocos: *router*, rotas e middleware. A funcionalidade principal de um servidor depende dos seus métodos de *routing*. Numa comunicação Cliente/Servidor, um cliente solicita alguns recursos do servidor, este irá localizá-los e responde com estes mesmos ao cliente. O Express facilita este trabalho, permitindo aos *developers* a criação de rotas numa estrutura simples [22].

Pedidos e Respostas

Uma aplicação Express utiliza funções *callback,* cujos parâmetros incluem o pedido e a resposta. O primeiro, o objeto do pedido, representa um pedido HTTP e possui propriedades para pesquisa, parâmetros, *body*, cabeçalhos HTTP, entre outros. O objeto da resposta representa a resposta HTTP, que a aplicação Express envia quando recebe um pedido HTTP [64].

Routing

Routing serve para determinar como uma aplicação responde a um pedido de um cliente num *endpoint/*rota específico, que consiste num Uniform Resource Identifier (URI) (ou *path*) e um método de pedido HTTP específico (*GET, POST, PUT* e *DELETE* são alguns dos mais utilizados) [64].

[55] https://www.paypal.com/pt/home
[56] https://uberportugal.pt/portugal/
[57] https://apiary.io/

Router

Uma rota básica no Express é criada da seguinte forma [22]:

```
app.Método (Caminho, MFunção)
```

- `app` é uma instância do Express;
- `Método` é o método do pedido HTTP;
- `Caminho` é precisamente o caminho da rota (URI);
- `MFunção` é a função do *middleware* que irá ser executada caso a rota corresponda à do pedido.

Middleware

O *middleware* são as funções que correspondem ao padrão [22]:

```
function(req, res, next)
```

- `Req` é o pedido do cliente;
- `Res` é a resposta do servidor;
- `Next` é a função *callback*.

Assim sendo, as funções do *middleware* executam qualquer código dentro delas, manipulando os objetos dos pedidos e das respostas.

3.3.3.1 Vantagens

Algumas das principais vantagens do Express.js [65]:

- Proporciona um desenvolvimento mais rápido e acessível de aplicações *web* Node.js.
- Simples de configurar e personalizar.
- Permite definir rotas da aplicação com base nos métodos HTTP e URLs.
- Inclui diversos módulos de *middleware* que podem ser utilizados para executar tarefas adicionais sob os pedidos e respostas.
- Simples de integrar com diferentes mecanismos de *templates*, nomeadamente o Jade[58], Vash[59], Embedded JavaScript templating (EJS)[60].
- Permite definir um *middleware* para lidar com erros.
- Facilidade em servir ficheiros estáticos e recursos da aplicação.
- Permite criar um servidor RESTful.
- Facilidade na conexão com bases de dados como o MongoDB, Redis[61] e MySQL.

[58] http://jade-lang.com/
[59] https://github.com/kirbysayshi/vash
[60] http://ejs.co/
[61] https://redis.io/

3.3.4 Angular 2

Angular é uma plataforma que facilita o desenvolvimento do *front-end* de aplicações *web* em HTML e utiliza uma linguagem que compila JavaScript, o TypeScript. Combina *templates* declarativos, injeção de dependência, ferramentas *end-to-end* (testam a funcionalidade e desempenho com cenários do mundo real [67]) e práticas recomendadas integradas para resolver desafios de desenvolvimento. Angular permite que os *developers* criem aplicações que estejam alocadas na *web*, em dispositivos móveis ou na área de trabalho [68].

AngularJS tem-se tornado uma *framework* fundamental na escolha de muitos *developers front-end*. O conceito AngularJS foi introduzido em 2009 por Misko Hevey e Adam Abrons, mas hoje em dia é gerido pela Google. A versão inicial 1.0 do AngularJS foi lançada em 2012. Foi desenvolvido para superar a vulnerabilidade e problemas de aplicações *web,* como por exemplo a nível da correção de *bugs*, segurança e desenvolver *interface*s de utilizadores de alta qualidade [39].

3.3.4.1 TypeScript

Criada pela Microsoft, TypeScript é uma linguagem de programação *open-source*. Não se trata de uma linguagem completamente nova, mas sim de um *superset* (ou superconjunto) do JavaScript. É muito semelhante às linguagens de programação modernas orientada a objetos como o C#, C++ ou Java, e tem como base quatro princípios fundamentais: encapsulamento, herança, abstração e polimorfismo. A Programação Orientada a Objetos (POO) sempre foi um problema quando aplicada em JavaScript, devido à sua sintaxe não possibilitar escrever classes, por exemplo. Adicionando funcionalidades que quando compiladas resultam em JavaScript novamente, o TypeScript disponibiliza uma forma de corrigir ou contornar esses problemas. É utilizada para desenvolver aplicações *web*

JavaScript tanto do lado do cliente como do servidor. Embora as aplicações em Angular 2 possam ser desenvolvidas utilizando ECMAScript(ES) 5, ES6 e dart[62], TypeScript disponibiliza uma ampla gama de recursos quando se trata de uma grande aplicação. A principal razão por trás da utilização desta linguagem é devido ao facto de esta proporcionar ótimas ferramentas como: refatoração (alterações na estrutura interna do código sem modificar o seu comportamento observável, de modo a torná-lo mais percetível) de código, preenchimento automático/*autocomplete*, navegação, escrita estática e um gerador de sintaxe completo [39].

Devido ao facto de ser um superconjunto do JavaScript, qualquer código elaborado com essa linguagem por ser colocado num arquivo TypeScript, que possua a extensão *".ts",* e utilizado diretamente. Este ponto é bastante positivo, uma vez que se pode utilizar códigos JavaScript já existentes, sem a necessidade de realizar grandes conversões. A

[62] https://webdev.dartlang.org/angular

criação de um projeto Angular 2 começa com a instalação do TypeScript no terminal, através do NPM, que gere todos os pacotes da linguagem [39].

3.3.4.2 AngularJS vs Angular 2

Após aproximadamente dois anos de pesquisa, a mesma equipa que produziu o AngularJS, lançou a versão final do Angular 2 em 2016. Este último pretende fornecer as melhores funcionalidades de renderização do lado do servidor para uma UI perfeita. Existem diversos problemas superados pelo Angular 2 sobre o AngularJS como [39]:

- Angular 2 fornece um desempenho muito superior ao AngularJS. As aplicações em Angular 2 colocam uma pressão mínima nas máquinas virtuais devido ao uso de pouca memória. Além disto, este fornece funcionalidades de reutilização de *templates* superior e de visualização da *cache*, entre outras que proporcionam um melhor desempenho.

- O Angular 2 possui recursos de *templates* mais avançados e poderosos. Os ambientes de desenvolvimento e os Integrated Development Environments (IDEs) são mais modernos e superiores ao tempo de execução do AngularJS. Isto auxilia os criadores de *templates* a descobrir os erros mais rápido, fornecendo assim mais tempo para o processo de escrita do *template* em si.

- Está mais preparado para futuros trabalhos em comparação ao AngularJS, dado que as tecnologias *web* são mais focadas na captação de live data, do que em *web crawling* (motores de busca utilizam rastreadores da rede para manter a base de dados atualizada). O Angular 2 possui como objetivo atender às necessidades permitindo uma suave renderização do lado do servidor para a *interface* do utilizador.

- O Angular 2 possui um suporte de recursos para aplicações móveis em plataformas *web*. A *framework* da UI móvel nativa do Angular 2 ajuda a tornar a *interface* do utilizador móvel totalmente funcional, tanto em plataformas IOS como Android.

3.3.4.3 Arquitetura

Aplicações desenvolvidas em Angular 2 encontram-se separadas em três partes: *templates* HTML, classes (plano a partir do qual objetos são criados [69]) de componentes que gerem os *templates*, e os serviços que contêm a lógica do negócio da aplicação. Esta abordagem concede um desenvolvimento rápido e lógico de aplicações, e facilita os testes. Numa aplicação Angular 2 corretamente projetada, cada parte não deve estar ciente dos detalhes

de implementação das outras partes [70]. Na Figura 10Figura 10 é possível observar como todas as partes fundamentais do Angular 2 se relacionam.

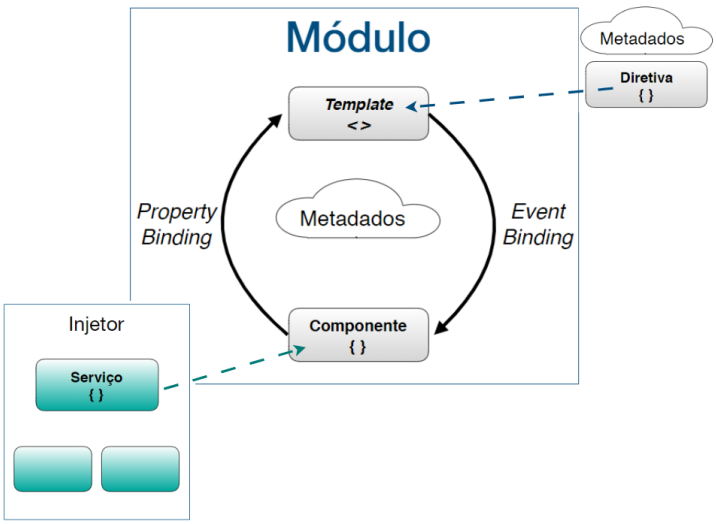

Figura 10 - Arquitetura do Angular 2. Adaptação: [71] .

Um componente em conjunto com o *template* forma uma *view* [72].

- Um *decorator* (fornece uma maneira de adicionar anotações e metadados à declaração de classes, métodos, propriedades ou parâmetros) numa classe de componente adiciona os metadados, assim como um indicador para o *template* associado.
- As diretivas e as marcas de *binding* no *template* de um componente alteram as *views* com base nos dados e na lógica do programa.

O injetor de dependência fornece os serviços a um componente [73].

Módulos

O bloco mais importante numa aplicação Angular 2 é o *root module*, que é utilizado para agrupar classes de componentes e serviços relacionados. Descreve como diferentes partes da aplicação são geradas simultaneamente. Todas as aplicações Angular 2 possuem um *root module*, convencionalmente denominado `AppModule`, que fornece um mecanismo *bootstrap* que inicia a aplicação. Tipicamente, os *route modules* são utilizados para criar uma aplicação simples, no entanto, também é possível utilizar *feature modules,* que estende as capacidades do outro. Qualquer um destes módulos é uma classe com um *decorator* designado `@NgModule` [39].

Componentes

Os componentes são a parte lógica de uma aplicação Angular 2 e controlam uma visão particular da ecrã denominada *view*. A lógica é definida dentro da classe do componente. Esta classe interage com a *view* através de uma API de propriedades e métodos. Cada componente possui um ciclo de vida, onde o Angular cria, atualiza, destrói e renderiza o

componente à medida que o utilizador se move na aplicação, e é possível realizar operações a cada momento neste ciclo de vida utilizando "ganchos" opcionais como `ngOnInit()` [72]. Esta função é executada quando o Angular inicia o componente.

Templates e Views

O *template* é uma forma de HTML que informa o Angular como deve renderizar o componente. As *views* tipicamente estão organizadas hierarquicamente, permitindo que se altere ou oculte/mostre seções da UI completas ou as páginas como uma unidade. Uma hierarquia de *views* (Figura 11Figura 11) tanto pode incluir *views* de componentes no mesmo ou em diferentes *NgModules* [72].

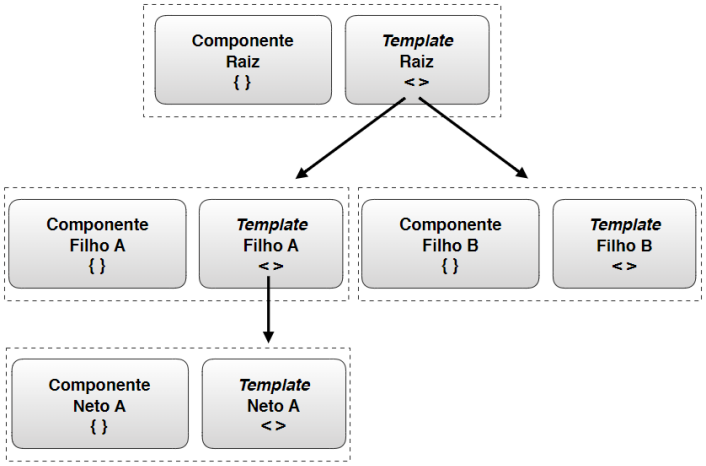

Figura 11 - Hierarquia de Views. Adaptação: [72].

Injeção de Dependência e Serviços

Um serviço é uma classe que serve para partilhar dados, que não estão associados a uma *view* específica, entre componentes. Realiza operações como a recolha de dados do servidor, validação de dados do utilizador e conexão direta à consola. Ao conceder estas tarefas a uma classe de um serviço, torna-as disponíveis para qualquer componente. A definição dos serviços é precedida imediatamente pelo *decorator* `@Injectable()`. Este disponibiliza os metadados, que por sua vez, permitem ao serviço ser injetado nos componentes do cliente como uma dependência [73].

O uso do *decorator* `@Injectable()` indica que o componente ou outra classe contém uma dependência. A injeção de dependência é utilizada no Angular 2 para fornecer serviços aos componentes que necessitem. Os componentes consomem serviços, ou seja, é possível injetar um serviço num componente, concedendo a este acesso à classe do serviço. A Figura 12 ilustra os componentes relativos à injeção de dependência no Angular 2 [73].

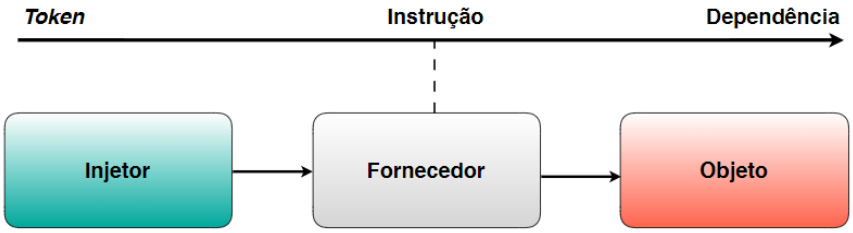

Figura 12 - Componentes da Injeção de Dependência. Adaptação: [39].

- Injetor: Expõe a API de modo a criar instâncias de dependência [39].
- Fornecedor: Aceita ajuda do injetor para criar instâncias de dependência [39].
- Dependência: É criado um objeto com dependências [39].

Data Binding e Metadados

Data binding é o processo de sincronização entre os dados e os diferentes componentes. O Angular 2 suporta *data binding* bidirecional, um mecanismo que realiza a coordenação entre os componentes e os *templates,* e desempenha também um papel fundamental na comunicação entre os componentes pai e filho. Atualiza os dados na *view*, adicionando a anotação no *template* HTML. A Figura 13 demonstra os quatro tipos de anotações possíveis, sendo que cada forma possui uma direção: para o DOM, do DOM ou ambos [72]. DOM é uma API que define a estrutura lógica dos documentos e como estes são acedidos e manipulados. Permite aos developers criar documentos XML ou HTML, navegar, modificar ou excluir elementos e conteúdo da sua estrutura [74].

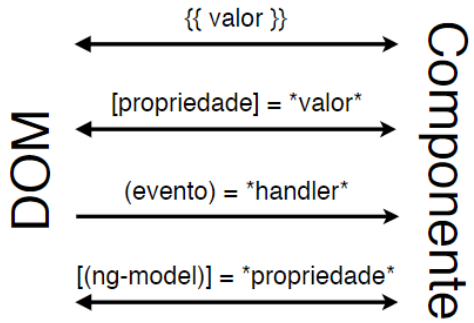

Figura 13 - Formas de data binding. Adaptação: [72].

Similarmente, outra ferramenta arquitetural também importante utilizada numa aplicação Angular 2 são os metadados. Estes são responsáveis pelo processamento das classes, ou seja, uma classe permanece uma classe e nunca se comporta como um componente por exemplo [39].

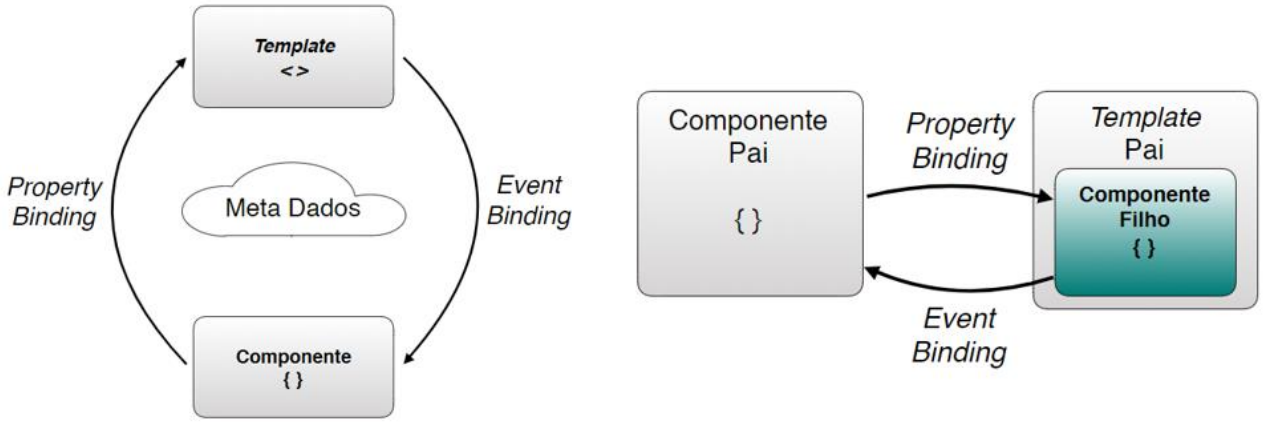

Figura 14 - Data binding na comunicação entre o template e o seu componente (esquerda), e entre os componentes pai e filho (direita). Adaptação: [72].

Diretivas e Componentes

Os *templates* em Angular são dinâmicos e quando são renderizados, transformam o DOM de acordo com as instruções fornecidas pelas diretivas. Uma diretiva é uma classe com o *decorator* `@Directive()` [72]. Existem três tipos de diretivas no Angular 2 [72]:

- Diretivas de atributo;
- Diretivas estruturais;
- Componentes.

Os componentes tecnicamente são uma diretiva, no entanto, são tão distintos e centrais para as aplicações Angular que são definidos com o *decorator* diferente `@Component()`, que estende o *decorator* `@Directive()` com características orientadas aos *templates*. Estes lidam com a parte do *templating* para criar a *interface* do utilizador [72].

As estruturais alteram a estrutura da *view* adicionando, removendo e substituindo elementos no objeto DOM, tornando assim o desenvolvimento da *interface* do utilizador mais simples e expressiva. Permitem por exemplo repetir uma coleção de objetos num documento ou ocultar/mostrar um elemento com base no valor da propriedade do componente [39].

As diretivas de atributos parecem atributos HTML normais e são utilizadas como atributos dos elementos. Desempenham um papel fundamental na alteração da aparência visual ou no comportamento dos componentes e dos elementos, afetando apenas os elementos onde se encontram. O exemplo mais notável destas diretivas é o `ngModel` [70]. Esta diretiva modifica o comportamento do elemento (`<input>`) definindo o valor da sua propriedade e respondendo à mudança de eventos [72].

O Angular possui mais diretivas pré-definidas que ou alteram a estrutura (`ngSwitch`) ou modificam aspetos dos elementos DOM e dos componentes (`ngStyle` ou `ngClass`) [72].

3.3.5 MVC

O *front-end* e o *back-end* são dois campos bem díspares, mas que se complementam na história do desenvolvimento *web*. No começo, estes dois campos emergiram com a necessidade de aprimorar as páginas *web* e assegurar-lhes mais funcionalidades, porém também causaram um aumento da complexidade do código. Posto isto, os *developers* especializavam-se apenas numa das duas categorias para responder melhor às necessidades de mercado. Desta forma, ao longo do tempo foram criadas *frameworks* para simplificar o desenvolvimento das páginas *web*, resultando assim na deslocação de funcionalidades, que anteriormente eram destinadas ao desenvolvimento *back-end*, para o *front-end*. Uma vez que a lógica de carregamento das páginas passou a ser realizada no cliente (navegador *web*) e não no servidor, houve uma diminuição no tempo de resposta do próprio servidor, e consequentemente, a diminuição do custo.

Nos dias que correm, o desenvolvimento *web* baseia-se num modelo de três camadas denominadas de dados, lógica e apresentação. Dados é a camada encarregada pelo armazenamento de dados; a camada da lógica é encarregue por estabelecer as regras de tratamento dos dados; e a apresentação é a camada que lida com a apresentação das páginas *web*. Estas camadas também podem ser subdivididas em base de dados, lógica do servidor, lógica do cliente e *interface* do cliente. Existem diversos padrões de desenvolvimento que adotam este modelo de três camadas, sendo um dos mais conhecidos e que é suportado pela MEAN *Stack* é o modelo MVC (Figura 15). O paradigma MVC distribui as três camadas em três objetos: *View, Controller* e *Model* [44]:

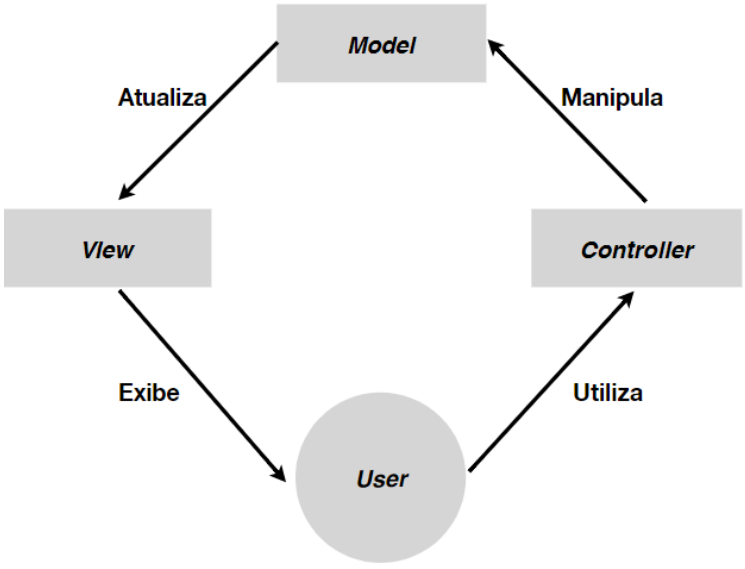

Figura 15 - Modelo MVC. Adaptação: [44].

- *View*: Cuida da interação do utilizador, isto é, a parte visual da aplicação.
- *Controller*: Comanda a *View* e o *Model* de acordo com os eventos do sistema, ou seja, a parte da lógica.
- *Model*: Manipula e armazena os dados.

Mais à frente (secção 4.3.4) será apresentado como este modelo MVC foi aplicado na solução da aplicação presente nesta dissertação.

3.4 RESTful API

Um dos fatores atraentes na utilização da MEAN *Stack* para conceber aplicações *web* é, o desenvolvimento de servidores RESTful. A criação destes servidores tornou-se uma tarefa muito importante e comum neste tipo de arquitetura, já que as aplicações necessitam cada vez mais de oferecer suporte para uma variedade de dispositivos, como telemóveis ou *tablets* [75].

Uma API é um conjunto de rotinas, protocolos, instruções e padrões para aceder a uma aplicação *web*. É a camada da aplicação que possui a oportunidade de interagir com outras aplicações. Para conseguir pedir dados de uma aplicação externa, por exemplo, é necessário criar um pedido no formato correto e enviá-lo para a API da aplicação de onde se pretende obter os dados. Esta por sua vez, lê o pedido, requisita os dados e cria uma resposta para enviar à aplicação que elaborou o pedido [76].

REST é um estilo arquitetural para aplicações de rede, que impõe um conjunto de restrições, baseado em padrões *web* que utilizam o protocolo HTTP. Foi proposto pela primeira vez em 2000, na tese de doutoramento do Dr. Roy Thomas Fielding, onde especificou uma arquitetura capaz de implementar serviços *web*, com o propósito de melhorar a visibilidade, fiabilidade e escalabilidade deste tipo de serviços. Qualquer serviço *web* ou API projetados com a arquitetura REST são denominados de Restful [77].

Na MEAN *Stack* a RESTful API é escrita em Node.js, utilizando a *framework* Express, permitindo assim realizar operações Create, Read, Update e Delete (CRUD) sobre a base de dados MongoDB (Figura 16) [75].

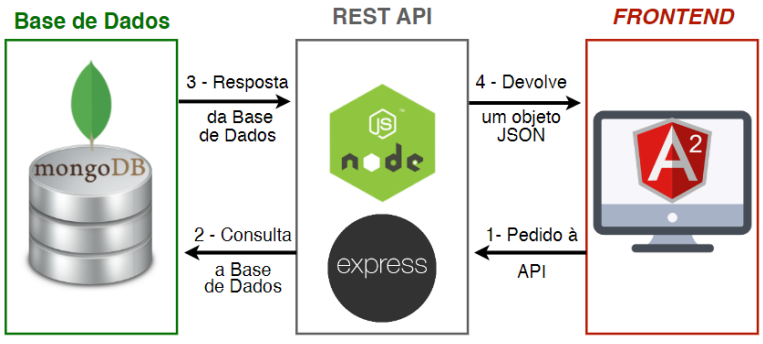

Figura 16 - REST API na arquitetura MEAN Stack. Adaptação: [78].

3.4.1 Restrições

Existem seis restrições principais que devem ser consideradas quando se pretende implementar uma RESTful API [77]:

1. Arquitetura cliente-servidor;
2. *Stateless* (desprovido de estado);
3. *Cache*;
4. *Interface* uniforme;
5. Sistema em camadas;
6. Código sob demanda.

Arquitetura cliente-servidor

Esta arquitetura é baseada no modelo cliente-servidor, onde o principal objetivo é separar as tarefas/responsabilidades entre estes dois componentes. Esta divisão possibilita o aumento da escalabilidade do servidor, promove incrementos em termos de segurança e facilita a portabilidade da *interface* do utilizador para diferentes plataformas. Alivia o cliente de tarefas como a comunicação com a base de dados, gestão da *cache*, entre outras. O contrário também é válido, dado que o servidor fica isento de encargos com a *interface* ou a experiência do utilizador. No entanto restringe as funcionalidades por parte do cliente, uma vez que concentra a carga de processamento no servidor [77].

Stateless

O segundo princípio chave dos seis apresentados indica que a comunicação não necessita de manter o estado da ligação ao longo de uma sessão. Por outras palavras, em cada pedido do utilizador é fundamental que toda a informação necessária para o servidor seja enviada, para que este processe o pedido sem ser obrigado a tirar partido de qualquer informação gravada anteriormente. Um servidor *stateless* não possui qualquer conhecimento sobre as aplicações conectadas a ele, dos conteúdos dos dados destas ou da forma como elas executam as suas funções. Assim cada interação nova é executada de forma independente das demais, ou seja, o servidor é autónomo do pedido seguinte, assim como do precedente. Este princípio valida diversas propriedades apresentados na Tabela 12 [77].

Tabela 12 - Propriedades do princípio stateless em REST [79].

Propriedades	Definição
Visibilidade	A visibilidade é melhorada, dado que este conceito possibilita que o sistema de monitorização não necessite de percorrer mais do que um ponto de referência, com a finalidade de determinar a natureza completa do pedido.
Fiabilidade	A fiabilidade é melhorada pois a tarefa de recuperação após a ocorrência de falhas parciais é simplificada.
Escalabilidade	A escalabilidade é melhorada dado que não é necessário armazenar o estado entre os diferentes pedidos, o servidor consegue dispensar recursos durante o seu funcionamento, simplificando assim o desenvolvimento do mesmo.

Cache

Como muitos clientes acedem ao mesmo servidor, e solicitam os mesmos recursos, é necessários que as respostas a estes possam ser guardadas na *cache* (cache local, proxy cache ou reverse proxy), de modo a evitar um processamento desnecessário e, por consequência, aumentar o desempenho. Por outras palavras, caso um cliente requisite um determinado recurso ao servidor, este último processa o pedido e armazena temporariamente na *cache*. Quando outros clientes efetuarem o mesmo pedido, o servidor já irá retornar o que se encontra na *cache*, sem necessitar de processar o requisito novamente [80].

Interface Uniforme

Uma característica do REST é a adoção de uma *interface* uniforme entre os componentes e é constituída por quatro restrições [80]:

- Identificação de recursos;
- Manipulação de recursos através de representações;
- Mensagens auto descritivas;
- Hipermídia como o motor de estado da aplicação (HATEOS).

O nome recurso em REST dá-se à abstração de informação e qualquer tipo de informação pode ser vista como um recurso, nomeadamente, imagens, documentos ou serviços. A identificação dos recursos é realizada por um URI, que reconhece um recurso pelo seu ID único e global, sendo a maioria das vezes criado por forma a ser legível pelos utilizadores. Os recursos são acedidos através dos URLs compostos por: nome do protocolo (normalmente HTTP), o Domain Name System (DNS) da máquina em que o recurso está localizado e o nome do arquivo que contém o recurso. Uma vez identificado o URL, o acesso é realizado através do envio do método HTTP e, dependendo do caso, através dos parâmetros inerentes àquela operação, definindo assim o pedido desejado. Após a execução de um determinado pedido para um URL que possui um determinado URI, a resposta pode ser apresentada de formas distintas: o utilizador dispõe da possibilidade de escolher a representação mais conveniente e que lhe facilite a gestão dos dados que recebe. Existem diversos tipos de representação, onde os mais comuns são o XML e JSON [77]. Ao prosseguir a comunicação entre os componentes por intermédio de uma representação dinâmica de recursos, o REST oculta a verdadeira natureza dos dados no remetente, sem restringir as funcionalidades do destinatário, concedendo a este trabalhar somente com as partes da representação que são relevantes para ele. Relativamente às mensagens auto descritivas, uma representação é um conjunto de dados que se associam aos metadados e que os detalham, de forma auto descritiva, com pares nome-valor. Uma vez que as representações REST não possuem um padrão universal, adequam-se às necessidades de cada projeto, permitindo assim aos *developers* uma maior flexibilidade para a definição de estruturas de representações. No que se refere ao HATEOS, este atua como um motor de navegação, onde *websites* orientados a hipermídia fornecem informações para navegar dinamicamente pela *interface* REST. A contribuição que a tecnologia HATEOS, concede o retorno de todas as informações necessárias na resposta, para que o cliente saiba navegar e tenha acesso a todos os recursos da aplicação [80].

Sistema em Camadas

No que se refere à quinta restrição, a aplicação deve ser composta por camadas, as quais devem ser fáceis de alterar, quer seja para aumentar o seu número, quer seja para diminuir. Este princípio sugere que o cliente nunca se deve ligar diretamente ao servidor, sem antes passar por um intermediário, que pode ser um *load balancer* ou outra máquina. Isto assegura que o cliente apenas se foque na comunicação com o intermediário e este, por sua vez, fica responsável por distribuir os pedidos pelos servidores [81].

Código sob demanda

Por fim, a restrição seis é opcional e permite que o cliente possa executar algum código sob demanda. Isto é, estender parte da lógica do servidor para o cliente. Possibilita que diferentes clientes possam comportar-se de formas especificas mesmo que utilizem os mesmos serviços, oferecidos pelos mesmo servidor [81].

3.4.2 Protocolo HTTP

Um dos princípios fundamentais da arquitetura REST é o estabelecimento de uma *interface* uniforme entre o cliente e o servidor. A utilização de uma *interface* uniforme proporciona uma arquitetura simplificada e desacoplada. Para tal, é necessário respeitar a semântica do protocolo utilizado pelos *web services*, o HTTP [80].

O HTTP é um protocolo de comunicação na camada de aplicação do modelo OSI, utilizado para trocar/transferir hipertexto na rede mundial de computadores, a World Wide Web. Hipertexto é um texto que contém links para outros textos, isto é, fornece informações como nós e redes de *links*, formando assim caminhos navegáveis que podem ser percorridos, retornados e referenciados. O HTTP é a base para a troca de dados pela Internet e foi projetado para ocultar detalhes da implementação dos serviços, apresentando uma *interface* uniforme para clientes realizarem pedidos, independentemente dos recursos associados ao serviço [82]. Para além disto, também foi projetado para funcionar como um protocolo intermediário para traduzir a comunicação de e para sistemas de informação não-HTTP. Os *proxies* (programa intermediária que opera como cliente e servidor para efetuar pedidos em nome de outros clientes [83]) e os *gateways* (servidor que atua como intermediário para outro servidor [83]) HTTP podem fornecer acesso a serviços de informação alternativos, convertendo os seus protocolos num formato de hipertexto que pode ser visualizado e manipulado pelos clientes da mesma forma que os serviços HTTP [84].

Este protocolo funciona como um modelo simples de pedido-resposta num modelo cliente-servidor. Aplicando regras simples, este protocolo descreve como as transações ao longo da Internet devem ser realizadas. Sempre que um servidor recebe um pedido HTTP, espera-se que este retorne uma resposta. Esta resposta é composta por uma lista de cabeçalhos e o corpo da mensagem [82].

3.4.2.1 Funcionamento

Para ser capaz de transferir dados ao longo da *web*, o protocolo HTTP depende dos protocolos TCP e IP. Estes últimos devem tornar possível a conexão entre o cliente e os servidores através de *sockets* TCP/IP. Um cliente é um programa que estabelece uma conexão com um servidor, com o propósito de enviar um ou mais pedidos HTTP. Um servidor é um programa que aceita conexões para responder a pedidos HTTP. Estes dois termos referem-se apenas às funções que os programas estão a exercer numa conexão especifica pois, o mesmo programa pode ser o cliente numa conexão e o servidor em outra [85].

O funcionamento do HTTP, processa-se da seguinte forma: o cliente inicia um pedido na forma de uma mensagem, através de uma conexão TCP para uma porta específica de um servidor. Este por sua vez, ao receber o pedido, responde com uma ou mais mensagens de resposta HTTP. O corpo desta mensagem é geralmente o recurso pedido. Os recursos HTTP são identificados e localizados na rede através do URL, utilizando os esquemas URI HTTP ou Hyper Text Transfer Protocol Secure (HTTPS) para indicar o uso de conexões inseguras ou seguras, respetivamente [84].

3.4.2.2 Mensagens HTTP

Como já foi referido anteriormente, a comunicação entre o cliente e o servidor é realizada através de mensagens. Existem dois tipos de mensagens: de pedidos que são enviadas pelos clientes e de resposta que são enviadas pelo servidor. Utilizam um formato genérico, definido na Request For Comments (RFC) 882[63]. Uma mensagem, independentemente se é de pedido ou resposta, possui uma estrutura semelhante e são compostas por [86]:

1. Uma *start-line* que descreve os pedidos a serem implementados ou o seu estado (sucesso ou falha). Sintaticamente, os dois tipos de mensagem diferem na *start-line*, sendo para os pedidos denomina-se *request-line* e para as respostas *status-line.* Diferem ainda no algoritmo para determinar o tamanho do corpo da mensagem.
    a. A *request-line* começa com um método *token* (como GET, PUT ou POST), seguido por um espaço único, o destino do pedido (normalmente um URL), outro espaço único, a versão do protocolo e termina com Carriage Return, Line Feed (CRLF).
    b. A *status-line* consiste na versão do protocolo, um espaço único, o código de *status*, outro único espaço, uma frase possivelmente vazia a descrever o código de *status* e termina também com CRLF.
2. Um conjunto opcional de cabeçalhos HTTP a especificar o pedido ou a descrever o corpo incluído na mensagem. Existem quatro tipos de

[63] https://tools.ietf.org/html/rfc882

cabeçalhos: *general-header*, *request-header*, *response-header* e *entity-header*.

3. Uma linha em branco indicando todos os metadados para o pedido.
4. Um corpo de mensagem opcional, que detém dados associados ao pedido (como o conteúdo de um formulário HTML) ou o documento associado a uma resposta.

A Figura 17 demonstra um exemplo de uma mensagem de pedido de um cliente, que pretende abrir a página `home.html` do *site* `xyz.com`, e a respetiva resposta do servidor a esse pedido.

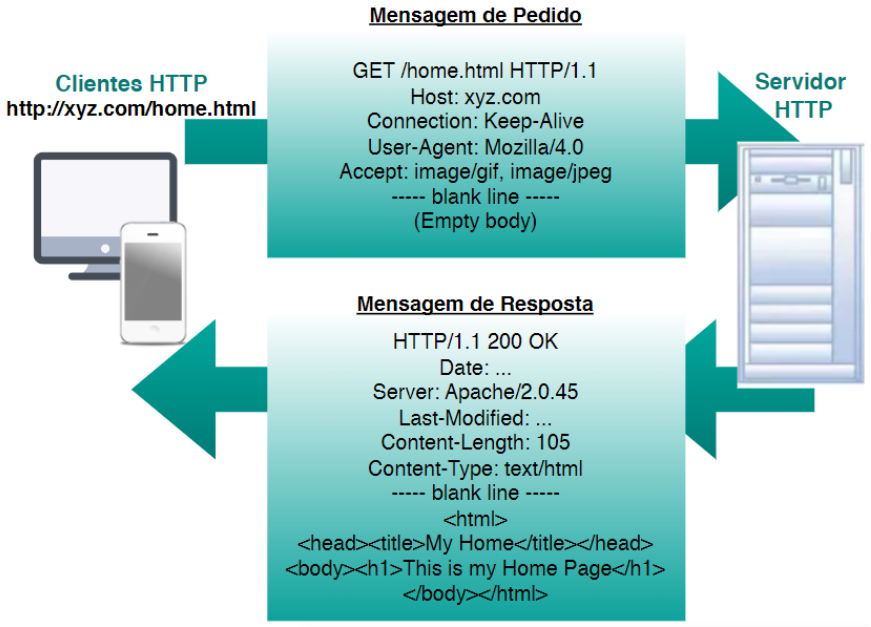

Figura 17 - Comunicação entre clientes e servidor com troca de mensagens. Adaptação:[87]

3.4.2.3 Métodos e Códigos de *Status*

O conjunto de métodos mais comuns utilizados no protocolo HTTP/1.1 está definido na Tabela 13 e pode ser expandido com base nos requisitos. São *case sensitive* e devem ser utilizados em letra maiúscula.

Tabela 13 - Métodos HTTP [83].

Método	Descrição
GET	Aplicado para recuperar informação de uma dado servidor utilizando um dado URI.
POST	É utilizado para enviar dados ao servidor, por exemplo informação do cliente ou ficheiros, através de formulários HTML.
PUT	Substitui todas as representações atuais do recurso destino com o conteúdo carregado.
DELETE	Remove todas as representações atuais do recurso destino por um URI.

O código de status é extremamente importante, dado que irá determinar como o cliente lida com a resposta. Está incluído na resposta do servidor e é um número inteiro de três dígitos, onde o primeiro define a classe da resposta e os dois últimos não possuem nenhuma função de categorização. Estes códigos são extensíveis e as aplicações HTTP não necessitam de entender o significado de todos os códigos de status registados [83]. Os principais códigos podem ser vistos na Tabela 14

Tabela 14 - Códigos de Status [83].

Códigos de *Status*	Categoria	Descrição
1XX	Informações.	Revela o nível do protocolo de transferência.
2XX	Sucessos.	Indica que o pedido do cliente foi aceite com sucesso.
3XX	Redirecionamentos.	Indica que o cliente deve tomar uma ação adicional por forma a completar o pedido.
4XX	Erros causados pelo cliente.	Esta categoria de código de *status* de erro aponta o dedo ao cliente.
5XX	Erros causados no servidor.	O servidor assume responsabilidade por este código de *status* de erro

3.5 *Softwares* Complementares

Para o processo de desenvolvimento da aplicação *web* foram escolhidos estes dois *softwares* complementares: o NetBeans e o Visual Studio Code[64] (VS Code). Nos tópicos seguintes são apresentadas algumas breves características dos mesmos.

[64] https://code.visualstudio.com/

3.5.1 NetBeans IDE

O NetBeans IDE[65] é um ambiente de desenvolvimento integrado *open-source*, baseado em Java que permite aos *developers* criar aplicações *web*, móveis e *desktop*. É um IDE gratuito que suporta o desenvolvimento com múltiplas linguagens incluindo Java, CSS, PHP, HTML e C++. Foi criado por dois estudantes em 1996 na Faculdade de Matemática e Física em Praga com o nome Xelfi e hoje em dia pertence à Apache Software Foundation[66] [88].

Adota uma arquitetura modular com um grande conjunto de ferramentas que auxiliam todo o ciclo de desenvolvimento desde o início do projeto até à implementação da aplicação. A sua arquitetura indica ainda que este IDE pode executar perfeitamente tanto no Windows, como no OS, Linux ou outros sistemas operacionais baseados em UNIX. É o IDE oficial do Java 8, com analisadores de código poderosos, conversores e editores, para além de uma comunidade mundial de utilizadores e *developers* ativos. Abaixo encontram-se algumas das razões pelas quais este IDE é uma ótima escolha [89].

Edição de código rápida e inteligente

O editor do NetBeans recua linhas, combina palavras e parênteses e destaca o código fonte sintaticamente e semanticamente. Permite refatorar facilmente o código com uma variedade de ferramentas úteis e avançadas e ainda oferece modelos de código, dicas de codificação e geradores de código. Como o editor é extensível, é possível inserir *plugins* para suportar diversas linguagens, como [89]:

- Java
- C/C++
- XML
- HTML
- PHP
- Groovy[67]
- JavaDoc[68]
- JavaScript
- Java Server Pages[69] (JSP)

A seguir encontram-se descritas algumas das vantagens deste IDE [89]:

Gestão simples e eficiente dos projetos

Proporciona uma visão geral clara de grandes aplicações, que contém milhares de pastas, arquivos e linhas de código. Tal é possível, devido ao facto do NetBeans fornecer diferentes

[65] https://netbeans.org/
[66] https://httpd.apache.org/
[67] http://groovy-lang.org/
[68] https://www.oracle.com/technetwork/articles/java/index-jsp-135444.html
[69] https://www.oracle.com/technetwork/java/index-jsp-138231.html

visões dos dados com múltiplas janelas de projetos e ferramentas úteis para configurar e gerir eficazmente as aplicações.

Desenvolvimento da interface do utilizador

Permite criar Graphical User *interface*s (GUIs) com facilidade e rapidez através da utilização de editores e de ferramentas *drag-and-drop* do IDE, para aplicações Java Platform Standard Edition (Java SE), HTML5[70], Java Platform Enterprise Edition (Java EE), PHP, C/C++ e Java Plataform Micro Edition (Java ME). Para as aplicações Java SE, o construtor de GUIs do NetBeans cuida automaticamente do espaçamento e do alinhamento. Este construtor é fácil de utilizar e intuitivo.

Código sem defeitos

O NetBeans fornece ferramentas de análise estática, especialmente com a FindBugs[71], para identificar e corrigir problemas comuns no código Java. Além disso, o *Debugger* permite colocar pontos de interrupção no código-fonte, percorrer o código, executar métodos, tirar *snapshots* e monitorizar a execução à medida que ela decorre. O Profiler do NetBeans concede assistência especializada para otimizar a utilização da velocidade e da memória da aplicação, facilitando a criação de aplicações fiáveis e escaláveis. Inclui ainda um *Debugger* visual próprio para aplicações Java SE, que possibilita depurar as *interface*s do utilizador sem consultar o código-fonte.

3.5.2 Visual Studio Code

O VS Code é um editor de código-fonte gratuito, *open-source* desenvolvido pela Microsoft que pode ser executado tanto no Windows, como no OS ou no Linux. É baseado na *framework* Electon[72] oferece suporte para a depuração, preenchimento de código inteligente, controlo do Git[73] integrado, destaques sintáticos, *snippets*, entre outras funcionalidades. A *interface* do utilizador é altamente personalizável, dado que os utilizadores podem alterar os temas, atalhos do teclado e outras preferências [90]. Algumas das vantagens/características deste editor são apresentadas abaixo [91]:

Suporte de linguagens

Para além do VS Code estar disponível em múltiplas plataformas, também dispõe de suporte para mais de trinta linguagens de programação. Fornece destaques sintáticos, correspondência de parênteses, assim como uma navegação no código simplificada. O seu objetivo prende-se em ser a primeira escolha dentro dos IDEs para o desenvolvimento de Node.js, ASP.NET[74] e TypeScript.

[70] https://www.w3.org/TR/html52/
[71] http://findbugs.sourceforge.net/
[72] https://electronjs.org/
[73] https://git-scm.com/
[74] https://www.asp.net/

Edição lado a lado

Possibilita um dos pedidos mais frequentes dos *developers* – editar diversas páginas de código apresentadas uma ao lado da outra. Suporta até três edições simultâneas, as quais também podem ser iniciadas por um comando *prompt*.

IntelliSense

O IntelliSense é um termo geral para vários recursos: Completar palavras, listar membros, informações sobre os parâmetros ou informação rápida. Estes recursos ajudam a aprender mais sobre o código que está a ser utilizado, a diminuir os erros e a manter uma melhor organização.

Espreitar Informação

Outra característica que o VS Code fornece é a possibilidade de observar o conteúdo de um ficheiro sem necessitar de abri-lo. Os resultados são embutidos na linha em que o programador se encontra e desaparecem ao clicar no botão *Escape*, poupando assim a mudança de ficheiro.

Controlo integrado da versão

O VS Code possui uma integração com o Git embutida e funciona com qualquer repositório do Git – local ou remoto. Consegue rastrear os ficheiros para alterações e oferece ações como *stage/unstage/commit*. Oferece dicas visuais para resolver conflitos antes dos *commits* do código.

CAPÍTULO 4

Desenvolvimento da Aplicação

Neste capítulo é detalhado todo o processo relativo ao desenvolvimento e implementação da aplicação *web*. Primeiro são identificados os requisitos necessários, que inclui os atores do sistema, um diagrama de *user stories* e os requisitos funcionais e não funcionais. De seguida, é ilustrada a arquitetura de toda a aplicação, acompanhada da sua interpretação. No final, são relatados ao detalhe todos os passos efetuados para a concretização da aplicação *web*, incluindo a explicação da extração dos dados, inserção na base de dados, da REST API, dos ficheiros alterados/criados e as suas funções.

4.1 Requisitos

4.1.1 Atores

Os utilizadores do sistema podem ser agrupados da seguinte forma:

• Administrador do sistema: único ator definido por omissão, tem acesso a todas as funcionalidades e vistas existentes no sistema.

• Utilizador Autorizado: utilizador definido pelo administrador do sistema. Pertence a um grupo de utilizadores com permissões de vistas e funcionalidades definidas.

4.1.2 User Stories

Cliente

A Figura 18 demonstra os *user stories* de um utilizador, que representam as funcionalidades presentes no sistema disponíveis a este. Neste caso um indivíduo é considerado um utilizador após ter efetuado o registo e respetivo *login*.

U.S.1 – Enquanto utilizador deve ser possível efetuar o registo e posteriormente o *login* na plataforma para aceder a funcionalidades exclusivas.

U.S.2 – Enquanto utilizador deve ser possível consultar as informações introduzidas no *login* e efetuar alterações sobre as mesmas.

U.S.3 – Enquanto utilizador deve existir a possibilidade de consultar todos os níveis da CIF.

U.S.4 – Enquanto utilizador deve ser admissível pesquisar sobre todos os campos da CIF para um rápido acesso a níveis mais profundos.

U.S.5 – Enquanto utilizador deve ser possível realizar propostas de exemplos para os campos inclui e exclui com a finalidade de construir uma classificação mais completa.

U.S.6 – Enquanto utilizador deve existir a possibilidade de criar um tópico de discussão no fórum, a fim de iniciar uma discussão relacionada com a CIF.

U.S.7 – Enquanto utilizador deve ser admissível comentar um tópico de discussão do fórum para responder a uma dada questão de outro utilizador ou proporcionar uma opinião sobre o assunto em questão.

U.S.8 – Enquanto utilizador deve ser possível pesquisar sobre todos os tópicos de discussão do fórum para um rápido acesso.

U.S.9 – Enquanto utilizador deve ser permissível consultar todas as propostas que foram submetidas pelo mesmo e os seus respetivos estados.

U.S.10 – Enquanto utilizador deve ser admissível consultar todas as propostas realizadas pelos outros utilizadores e os seus respetivos estados.

U.S.11 – Enquanto utilizador deve ser permitida a votação positiva ou negativa sobre propostas com o estado “por aprovar” submetidas por outros utilizados.

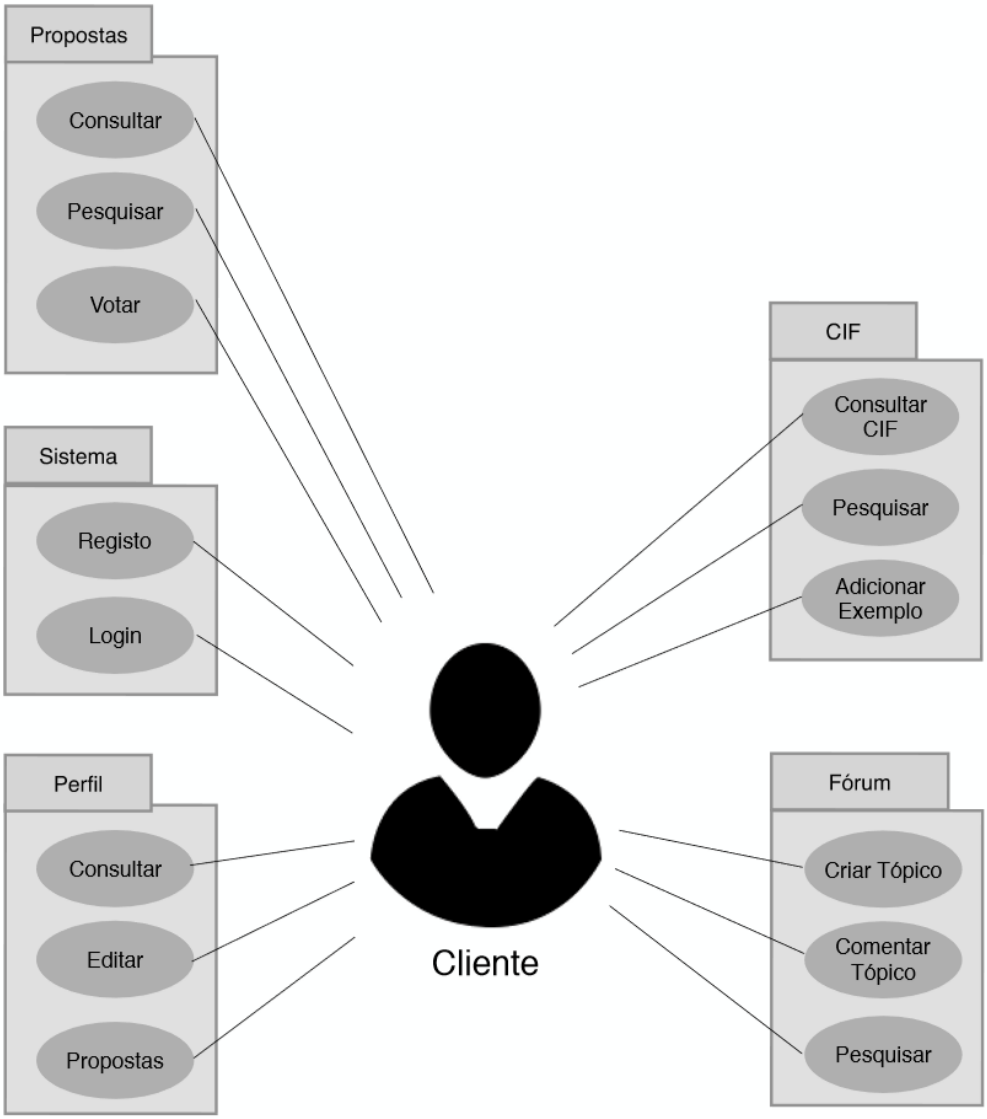

Figura 18 - User Stories do Cliente.

Administrador

A Figura 19 exibe o *user story* relativo ao administrador. Este dispõe de todas as funcionalidades do utilizador e ainda a gestão do sistema.

U.S.12 – Enquanto administrador deve ser possível efetuar o *login* na plataforma para ter acesso a todas as funcionalidades exclusivas ao seu estatuto.

U.S.13 – Enquanto administrador deve existir a possibilidade de aprovar ou rejeitar registos de utilizadores para determinar quem pode aceder à plataforma.

U.S.14 – Enquanto administrador deve ser permissível aprovar ou rejeitar propostas de utilizadores sobre as instâncias inclui e exclui da CIF, para adicionar ou não à versão disponibilizada a todos os utilizadores da plataforma.

U.S.15 – Enquanto administrador deve ser permitido aprovar ou rejeitar propostas de tópicos de discussão para serem adicionadas ao fórum disponível na plataforma.

U.S.16 – Enquanto administrador deve ser possível pesquisar sobre todos os utilizadores registados, para um rápido acesso às suas informações.

U.S.17 – Enquanto administrador deve existir a possibilidade de consultar e pesquisar sobre o histórico de todas as propostas realizadas às instâncias inclui e exclui da CIF.

U.S.18 – Enquanto administrador deve existir a possibilidade de consultar todos os níveis da CIF.

U.S.19 – Enquanto administrador deve ser admissível pesquisar sobre todos os campos da CIF para um rápido acesso a níveis mais profundos.

U.S.20 – Enquanto administrador deve ser possível adicionar exemplos nos campos inclui e exclui com a finalidade de construir uma classificação mais completa.

U.S.21 – Enquanto administrador deve existir a possibilidade de editar as instâncias de inclui e exclui da CIF a fim de, corrigir ou remover os exemplos já existentes.

U.S.22 – Enquanto administrador deve existir a possibilidade de criar um tópico de discussão no fórum, com o propósito de iniciar uma discussão relacionada com a CIF.

U.S.23 – Enquanto administrador deve ser admissível comentar um tópico de discussão do fórum, para responder a uma dada questão de outro utilizador ou proporcionar uma opinião sobre o assunto em questão.

U.S.24 – Enquanto administrador deve ser possível pesquisar sobre todos os tópicos de discussão do fórum para um rápido acesso.

U.S.25 – Enquanto administrador deve ser permissível a finalização de um tópico de discussão do fórum, com o propósito de impedir a introdução de novos comentários.

U.S.26 – Enquanto administrador é suposto ter acesso à opção de remover tópicos de discussão do fórum.

U.S.27 – Enquanto administrador deve ser possível consultar as informações introduzidas no *login* e efetuar alterações sobre as mesmas.

U.S.28 – Enquanto administrador deve ser admissível consultar todas as propostas realizadas pelos outros utilizadores e os seus respetivos estados.

U.S.29 – Enquanto administrador deve ser permitida a votação positiva ou negativa sobre propostas com o estado “por aprovar” submetidas por outros utilizados.

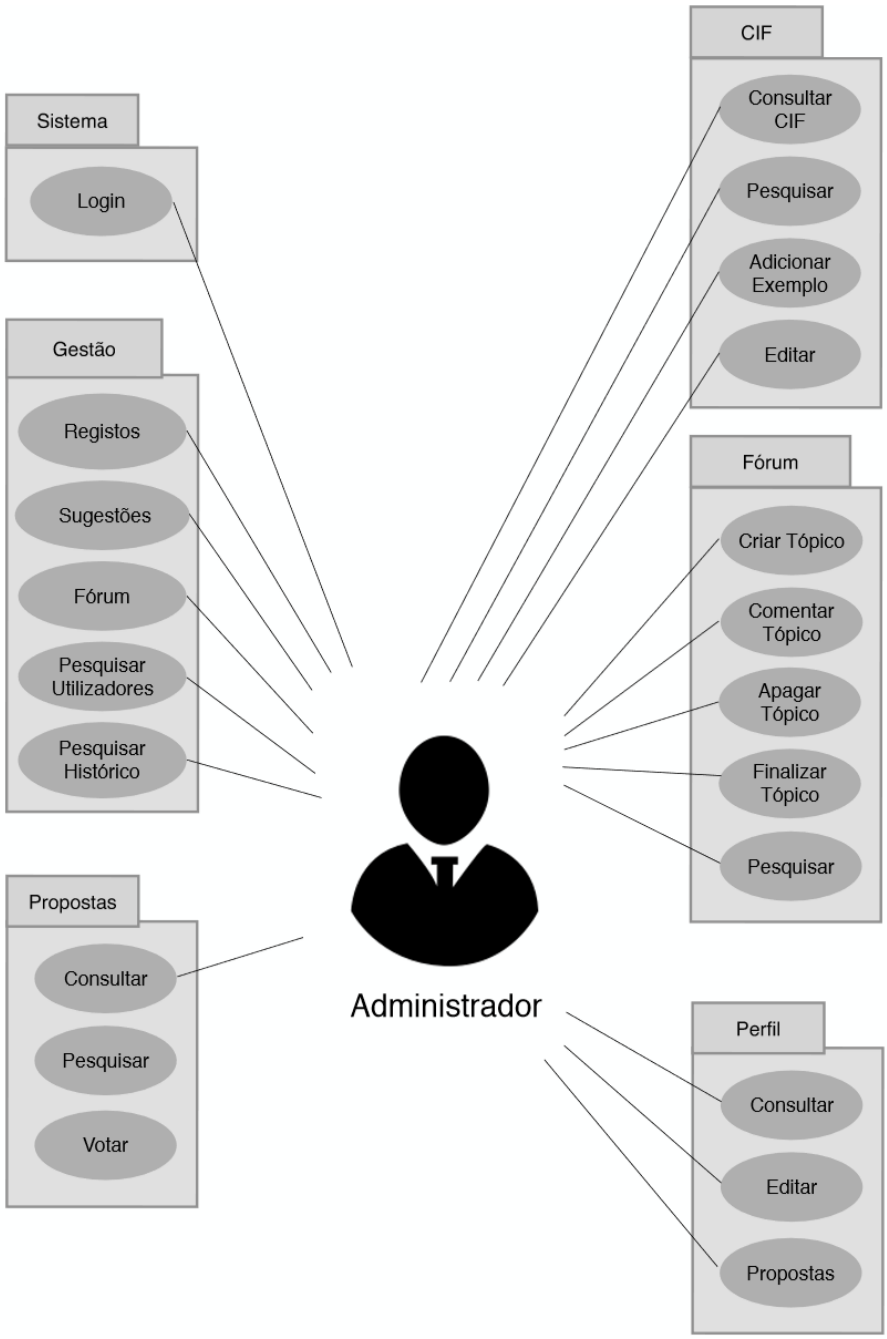

Figura 19 - User Stories do Administrador.

4.1.3 Requisitos Funcionais

Requisitos do sistema

- A aplicação deve suportar diferentes utilizadores, como o administrador e o utilizador comum.

- O sistema deve possibilitar o registo apenas a pessoas que possuem um número de identificação ou que descrevam num pequeno texto a intenção do seu registo.
- O sistema deve permitir o *login* de utilizadores com o registo aprovado pelo administrador.
- O sistema deve disponibilizar determinadas páginas e botões apenas a utilizadores com a sessão iniciada ou apenas ao administrador.
- O sistema deve autorizar comentários apenas nos tópicos disponíveis para discussão.
- O sistema deve permitir a votação positiva ou negativa apenas de propostas que se encontrem com o estado “por aprovar”.

Utilizador

- O sistema deve possibilitar a consulta, pesquisa e adição de exemplos e edição de exemplos nas instâncias inclui e exclui da CIF.
- O sistema deve conceber a pesquisa, comentários e criação de tópicos de discussão.
- O sistema deve permitir a consulta e edição do perfil.
- O sistema deve possibilitar a pesquisa de propostas e a votação apenas naquelas com o estado “por aprovar”.
- O sistema deve permitir a consulta das propostas, que foram submetidas pelo utilizador, e do seu respetivo estado.

Administrador

- O sistema deve permitir a aprovação ou rejeição de registos de utilizadores.
- O sistema deve permitir a aprovação ou rejeição de propostas de exemplos a adicionar nas instâncias inclui e exclui, realizadas por utilizadores.
- O sistema deve autorizar a aprovação ou rejeição de tópicos de discussão para o fórum.
- O sistema deve conceber a pesquisa de utilizadores registados.
- O sistema deve autorizar a consulta e pesquisa do histórico.
- O sistema deve possibilitar a consulta, pesquisa com filtros, adição e edição de exemplos da CIF.
- O sistema deve conceber a pesquisa, criação, remoção e finalização de tópicos de discussão.
- O sistema deve permitir a consulta e edição do perfil.
- O sistema deve conceber a pesquisa e consulta de todas as pesquisas submetidas e os seus respetivos estados.

4.1.4 Requisitos não funcionais

Os requisitos não funcionais foram categorizados de acordo com o modelo FURPS+ concebido por Robert Grady. O acrónimo representa “Functionality, Usability, Reliability, Performance and Supportability” e o + permite especificar as restrições incluindo o *Design*, Implementação, *Interface* e restrições físicas. Esta técnica fez com que a classificação de requisitos enfatizasse a compreensão dos diferentes tipos de requisitos não funcionais. A seguir encontram-se descritos os requisitos não funcionais referentes à aplicação *web* retratada neste projeto.

RNF001- O sistema deve ser de fácil aprendizagem (usabilidade).

RNF002- O sistema deve possuir uma *interface* (GUI) para a interação com o utilizador (Fatores Humanos).

RNF003- O sistema deve ter uma base de dados desenvolvida no SGBD MongoDB (Configurabilidade).

RNF004- O sistema deve adaptar-se às alterações de qualquer formato de entrada sem necessidade de recompilar o código (Configurabilidade).

RNF005- Um *developer* de *software* com seis meses de experiência deve conseguir corrigir qualquer defeito em menos de dois dias (Manutenção).

RNF006- Se o formato dos dados de entrada for alterado, o *developer* deve conseguir fazer as alterações necessárias em menos de vinte horas (Extensibilidade).

RNF007- 90% dos utilizadores inexperientes devem conseguir aprender a operar os principais casos de uso sem assistência externa (Compreensibilidade).

RNF008- O tempo de resposta do sistema não deve ultrapassar os dez segundos por pedido (Tempo de Resposta).

RNF008- O sistema deve executar em qualquer tipo de *software* (Portabilidade).

4.2 Arquitetura

A Figura 20 descreve a arquitetura da aplicação *web*, apresentando todos os seus componentes e as interações que ocorrem entre estes.

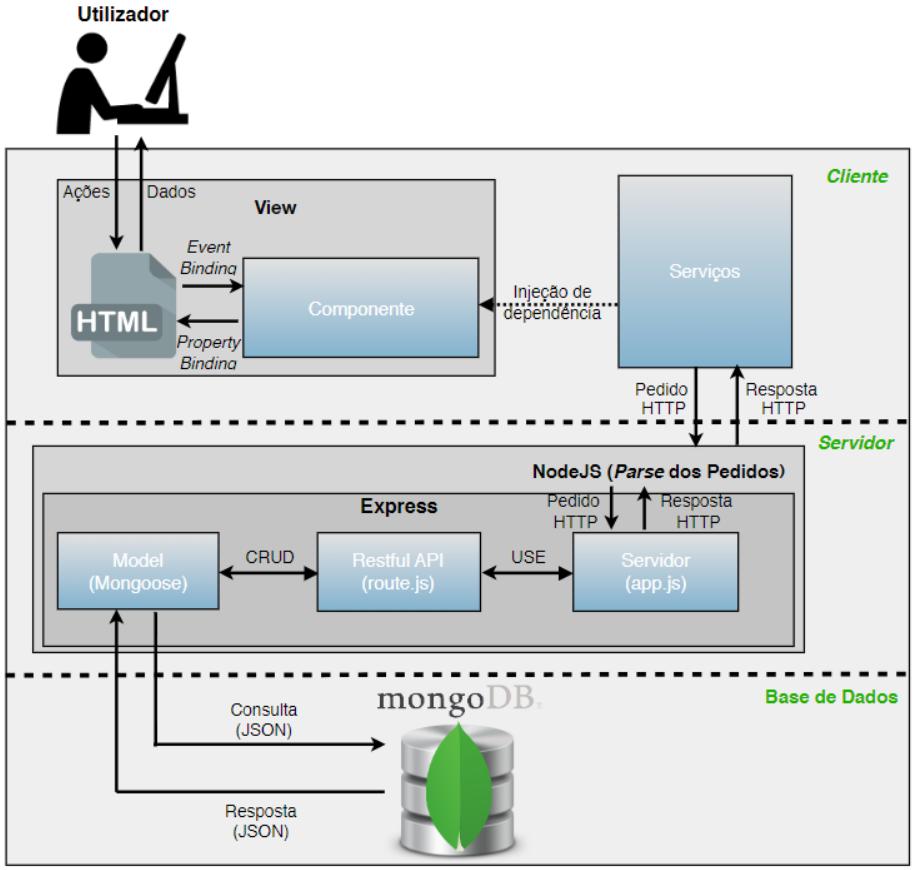

Figura 20 - Arquitetura do protótipo aplicação web.

Cliente

O principal meio de interação entre o utilizador e a solução final são as páginas *html*, que expressam e possibilitam a execução das ações que este efetua. Cada página *html* está associada a um componente, onde este por sua vez define uma classe que contém os dados e a lógica da aplicação. Quando o utilizador realiza uma operação no ficheiro *html*, despoleta uma função no seu componente correspondente, através de *event binding*. Se forem requeridos dados provenientes da base de dados é necessário invocar um Serviço, pois este funciona como um *wrapper* do lado do cliente para os *endpoints* da RESTful API. Os Componentes consomem os Serviços, ou seja, é possível injetar um Serviço num Componente, concedendo a este acesso à classe do Serviço. Estes são utilizados pelos componentes para enviar e receber dados da base de dados, onde através do *import* do protocolo HTTP, é possível realizar pedidos ao servidor. O pedido é realizado com uma instância *http*, que permite invocar os métodos GET, PUT, POST ou DELETE, passando um URL que será o *endpoint* e pode conter possíveis parâmetros em formato JSON. A

manipulação e tratamento de erros é realizada com a ajuda dos operadores observáveis *map* e *catch* que também são importados.

Servidor

O pedido proveniente do Serviço é inicialmente acedido pelo Node.js *threading* e posteriormente enviado para o servidor iniciado pelo Express. Esta última *framework* permite configurar a aplicação, adicionar rotas e *rest endpoints* para inicializar o *middleware*. No servidor é definido o caminho base para o ficheiro onde se encontram os *endpoints/*rotas da API com as operações CRUD à base de dados (`route.js`). Neste ficheiro é utilizada uma instância da classe Router do Express para definir rotas da RESTful API e é onde existem diversas funções constituídas por dois argumentos: uma rota e uma função *callback* (ver secção 3.3.3). Se essa rota corresponder ao URL enviado no pedido HTTP, então essa função será executada. A função *callback* contém os objetos do pedido e da resposta como argumentos. Dentro destas funções, são efetuadas operações CRUD à base de dados MongoDB, com o auxílio dos modelos de dados definidos pela biblioteca Mongoose. Os modelos de dados são schemas que mapeiam uma coleção do MongoDB e definem o formato dos documentos existentes nesta. As instâncias dos modelos são documentos, que dispõem de uma série de métodos incluídos (CREATE, SAVE, REMOVE e FIND) que permitem armazenar e recuperar dados da base de dados.

Base de dados

A resposta da base de dados faz o caminho inverso. Quando chega ao Serviço que realizou o pedido HTTP, com o operador *map*, é invocado o método *.json* para converter os dados para JSON e enviá-los para qualquer subescritor (Componente) que esteja à espera da resposta. É assim retornado um observável para o Componente, que para aceder aos dados da resposta necessita de subscrever a ele e atribuir os dados a uma variável. Esta variável pode ser acedida pela página HTML, através de *property binding* e disposta ao utilizador.

4.3 Implementação

4.3.1 Base de dados

O primeiro passo a ser tomado no desenvolvimento desta aplicação, foi a criação da base de dados MongoDB. Através do terminal, foi utilizado o comando `use cif`, que gera uma base de dados intitulada `cif`. De seguida, é criada a coleção designada `data`, utilizando o comando `cif.createCollection('data')`. Esta coleção irá conter os documentos com todas as informações relativas à CIF. A estrutura de cada documento difere consoante os dados disponíveis de cada código ou devido a alterações realizadas na aplicação sobre os mesmos.

4.3.1.1 Extração dos dados

O segundo passo foi a extração dos dados da CIF para um formato que possa ser utilizado e inserido numa base de dados. O Instituto Nacional para a Reabilitação[75] (INR) disponibilizou diversos ficheiros no formato HTML para descarregar no seu *website*. A partir destes duzentos e sessenta e cinco ficheiros foi desenvolvido um projeto Java, utilizando o IDE Netbeans, com o intuito de realizar a extração dos dados e seguidamente a inserção dos mesmos na base de dados MongoDB. Inicialmente foi efetuado um estudo dos ficheiros HTML e posteriormente desenvolvido um padrão para retirar todos os campos requeridos.

4.3.1.2 Projeto Java

O projeto denomina-se *mongo* e é constituído por três ficheiros:

- *Index.java*
- *Chapters.java*
- *Rest.java*

Cada um destes ficheiros trata de padrões HTML diferentes. O projeto contém a pasta `files` que por sua vez detém duas pastas: `chapters` e `rest`. A primeira possui apenas os ficheiros `d.html`, `b.html`, `s.html` e `e.html`, e é utilizada pelo ficheiro `Chapters.java`. A segunda dispõe do resto dos ficheiros disponibilizados pelo INR e são utilizados pelo ficheiro `Rest.java`.

4.3.1.3 Conexão à base de dados

No início de cada ficheiro Java existe um excerto de código (Figura 22) correspondente à conexão com a base de dados MongoDB criada previamente. Para tal foi utilizado o MongoDB Java driver 3.4.3[76].

```
MongoClient mongoClient = new MongoClient("localhost", 27017);
System.out.println("server connection successfully done");
DB database = mongoClient.getDB("cif");
System.out.println("Connection Done");
System.out.println("Database Name " + database.getName());

DBCollection coll = database.getCollection("data");

System.out.println("Collection created successfully");
```

Figura 21 - Excerto da conexão ao MongoDB.

Primeiro foi criado um objeto da classe MongoClient para ser possível conectar ao servidor MongoDB. Esta classe é segura para *threads* e é partilhada entre elas.

[75] http://www.inr.pt/
[76] https://mongodb.github.io/mongo-java-driver/

De seguida é realizada a conexão à base de dados, de nome `cif` e iniciado um objeto com a Coleção, também já gerada anteriormente, de nome `data`, onde os dados irão ser inseridos.

4.3.1.4 Tipos de ficheiros

Após um estudo sobre as diversas estruturas existentes nos ficheiros fornecidos pelo INR, foi possível observar os diferentes tipos existentes. Com base nisto, foram criados os três ficheiros do projeto que tratam de cada um destes padrões.

Index.java

O primeiro ficheiro, `Index.java`, consiste na inserção manual dos códigos `b`,`e`,`d` e `s`. Estes códigos contêm apenas três campos: código, nome e descrição (Figura 22).

```
String n1 = "Funções do Corpo";
String d1 = "Funções do corpo são as funções fisiológicas dos sistemas orgânicos (incluindo as fu
String c1 = "b";
BasicDBObject doc1 = new BasicDBObject("name", n1).append("code", c1).append("description", d1);
//insercao na base de dados
coll.insert(doc1);

String n2 = "Estruturas do Corpo";
String d2 = "As estruturas do corpo são partes anatómicas do corpo, tais como, órgãos, membros e
String c2 = "s";
BasicDBObject doc2 = new BasicDBObject("name", n2).append("code", c2).append("description", d2);
//insercao na base de dados
coll.insert(doc2);

String n3 = "Actividades e Participação";
String d3 = "Actividade é a execução de uma tarefa ou acção por um indivíduo. Participação é envo
String c3 = "d";
  BasicDBObject doc3 = new BasicDBObject("name", n3).append("code", c3).append("description", d3)
  //insercao na base de dados
coll.insert(doc3);

String n4 = "Factores Ambientais";
String d4 = "Os factores ambientais constituem o ambiente físico, social e atitudinal em que as p
String c4 = "e";
BasicDBObject doc4 = new BasicDBObject("name", n4).append("code", c4).append("description", d4);
//insercao na base de dados
coll.insert(doc4);
```

Figura 22 - Inserção manual dos códigos na base de dados.

Para cada código é criado um objeto da classe BasicDBObject, que corresponde a um documento onde são inseridas as *Strings* com os dados e o nome que se pretende dar ao campo.

Chapters.java

No ficheiro Chapters.java foi utilizada a biblioteca Jsoup[77], que fornece uma API conveniente para analisar, extrair e manipular dados armazenados em documentos HTML, utilizando métodos DOM, CSS e Jquery[78]. Este programa faz o *parse* apenas dos ficheiros `b.html`, `d.html`, `e.html` e `s.html`, que se encontram no diretório `files/chapters`.

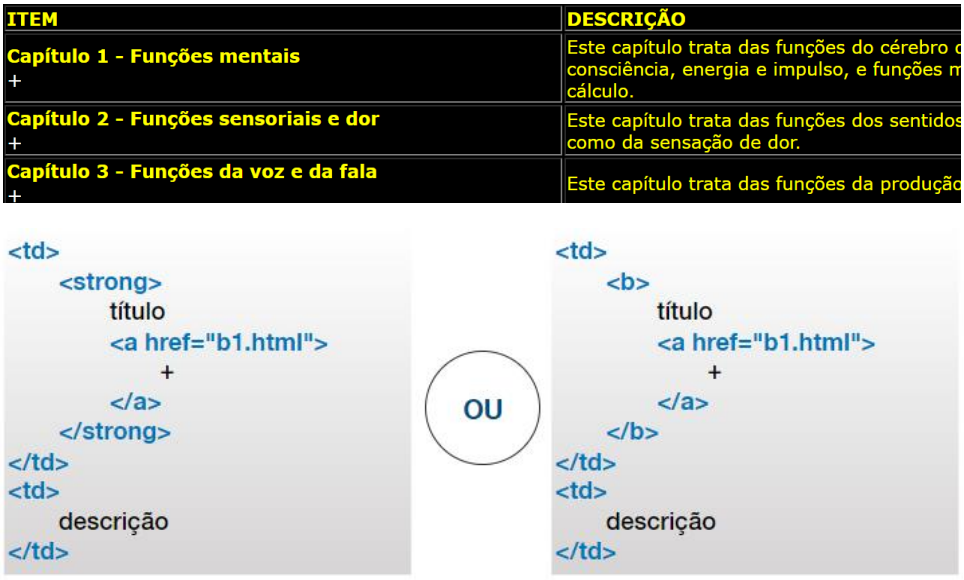

Figura 23 - Estrutura dos ficheiros relativos aos capítulos.

Existem dois tipos de estruturas nestes quatro ficheiros, como é possível observar na parte inferior da Figura 23. Na parte superior da imagem verificar-se o ficheiro `b.html`, onde cada linha apresentada corresponde a uma das estruturas da parte inferior da imagem. Para retirar o título é executado um ciclo *for* que percorre ou a *tag* `strong` (lado esquerdo) ou a *tag* `b` (lado direito), consoante a estrutura do ficheiro. O código em ambas as estruturas é a soma de um contador que aumenta com o ciclo e o nome do ficheiro. No exemplo da imagem, o ficheiro é o `b.html`, quando o ciclo se encontra na primeira linha (capítulo 1), então o código será b1, quando passa para a segunda linha (capítulo 2) o código é b2, e assim sucessivamente. De seguida, para recolher a descrição é necessário selecionar a *tag* `td`. No entanto a *tag* `td` que envolve o título também é selecionada, daí ser necessário retirar a *tag* `b/strong` e a *tag* `a`, dado que possuem informação, permanecendo no fim apenas a descrição.

O código e o título são inseridos num mapa e as descrições numa lista. No fim de ler todos os ficheiros, é percorrido o mapa com a lista das descrições, onde são associados num documento e por fim, este documento é inserido na coleção da base de dados.

[77] https://jsoup.org/
[78] https://jquery.com/

Rest.java

O último ficheiro do programa, `Rest.java`, é o mais complexo, dado que trata de diversas estruturas HTML distintas, onde por vezes o único ponto que difere é uma palavra que se encontra a negrito no centro do ficheiro, alterando assim uma *tag*. Os ficheiros lidos por este programa estão localizados no diretório `files/resto`. Todos contém duas *tags* em comum: a *tag* `tbody` e a *tag* `tr`. Com isto, é selecionado o último `tbody` do ficheiro e dentro deste as *tags* `tr`, que serão percorridas por um ciclo for. Dentro deste ciclo é verificado o tipo de estrutura.

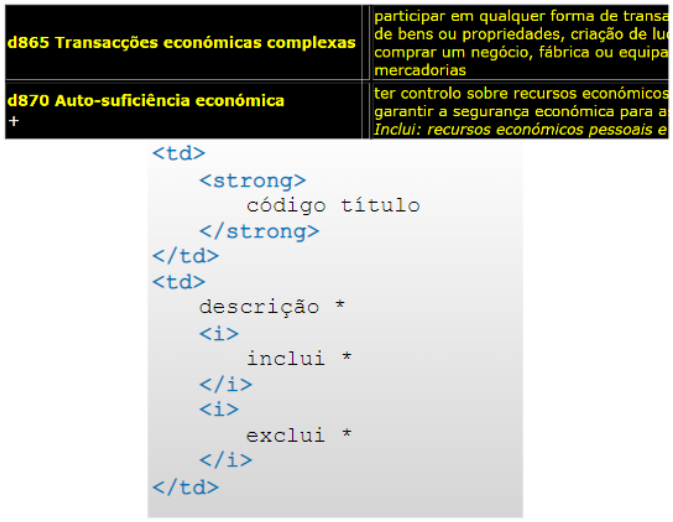

Figura 24 - Exemplo 1 da estrutura tratada no ficheiro Rest.java.

A parte inferior da Figura 24 apresenta a primeira estrutura analisada no programa, que correspondente a uma linha da parte superior da imagem. Os asteriscos assinalados indicam que o campo pode ou não existir, diferindo entre códigos e ficheiros. Esta estrutura distingue-se das outras devido a possuir o código e o título dentro de uma *tag* `strong`. Deste modo, é verificado se esta *tag* se encontra no ficheiro, caso exista então trata-se deste tipo de estrutura, onde o código e o título são guardados numa variável. De seguida, se existir a segunda *tag* `td`, indica a presença de uma descrição, que por sua vez pode conter o campo inclui ou exclui. Assim sendo, é necessário verificar estes três campos e consoante a sua existência é gerado um documento com campos diferentes.

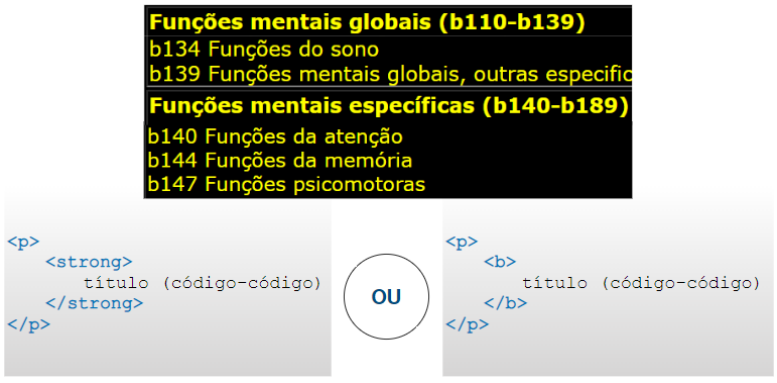

Figura 25 - Exemplo 2 da estrutura tratada no ficheiro Rest.java.

Caso não seja encontra a *tag* `strong`, é verificado se existe a *tag* `p`, que remete para o segundo tipo de estrutura. Estes ficheiros contém um intervalo de códigos como se pode observar na parte superior da Figura 25, isto é, não dispõe do mesmo sistema que os outros documentos código-título. Indica somente que os códigos b110 até ao b139 estão incluídos nas funções mentais globais por exemplo. Deste modo, quando se encontra a *tag* `p`, é examinado se existe a *tag* `strong` ou a *tag* `b` e dentro destas é retirado o título e o intervalo de códigos como se fosse um código normal.

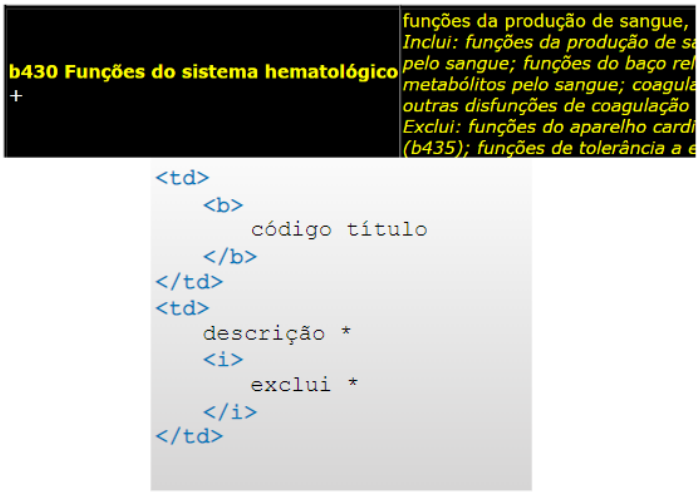

Figura 26 - Exemplo 3 da estrutura tratada no ficheiro Rest.java.

Por fim, quando não entra na verificação dos tipos de estruturas anteriores, então trata-se da disposição demonstrada na Figura 26. Inicialmente é retirado o código e o título da seleção da *tag* `b`. Após isto, é averiguada a existência da última *tag* `td`, confirmando se existe descrição, que por sua vez, obriga à análise dos campos inclui e exclui. A inserção dos dados na base de dados é efetuada sempre após a identificação do tipo de estrutura do ficheiro que está a ser lido.

4.3.1.5 Diagrama da Base de Dados

A base de dados é constituída por duas coleções: `data` e `users`. A última coleção é destinada apenas aos dados de utilizadores que se encontram registados no sistema, enquanto que a primeira, `data`, detém o resto dos dados todos encontrados na aplicação *web*. Na Figura 27 encontra-se ilustrado o diagrama da base de dados.

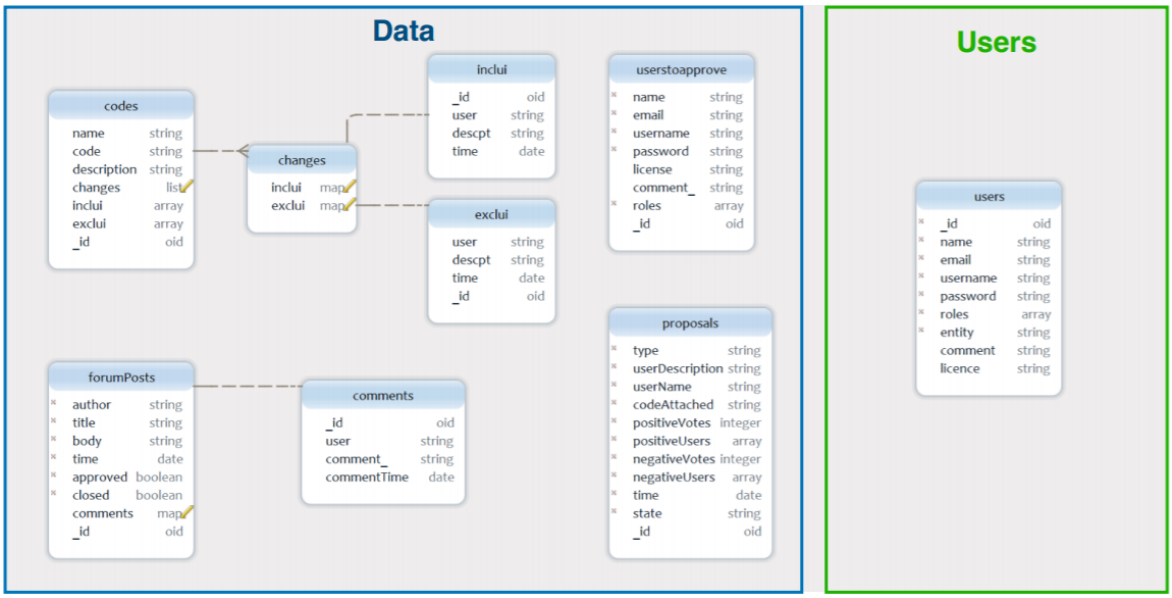

Figura 27 - Diagrama da Base de dados.

A tabela `codes` apresenta a estrutura de todos os códigos da CIF. No entanto apenas os campos `name`, `code` e `description` são obrigatórios. O campo `changes`, é um objeto com dois *arrays* de documentos, `inclui` e `exclui`. Este destina-se a guardar as alterações que ocorrem nos campos `inclui` e `exclui` e, portanto, a sua existência depende destes dois campos.

A tabela `forumPosts`, assim como o nome indica, refere-se aos dados dos tópicos de discussão criados na página do Fórum da aplicação. Os campos `approved` e `closed` por defeito, são iniciados com o valor *false*. O tópico apenas é disposto na página caso o administrador o tenha aprovado, ou seja, quando o campo `approved` se encontrar com o valor *true*. Caso o administrador, ache que o tópico em questão já não necessita de ser discutido, finaliza-o, isto é, o valor do campo `closed` passa a *true*.

`Proposals` é a tabela relativa a todas as propostas de exemplos submetidas pelos utilizadores. Para além das informações relativas à proposta, inclui votos positivos (`positiveVotes`), negativos (`negativeVotes`) e os seus respetivos utilizadores (`positiveUsers` e `NegativeUsers`). Detém ainda um estado (`state`) que pode ser: “aprovado”, “rejeitado” ou “por aprovar” que é determinado pela ação do administrador.

Por fim, a tabela `userstoapprove` pertence aos dados dos utilizadores, os quais o administrador ainda não aprovou o seu registo na plataforma. Uma vez aprovados, estes dados são incluídos na coleção `users` (tabela `users`) e apagados da coleção atual (`data`).

Um exemplo de uma estrutura de um documento é possível observar na Figura 28, que detém informações sobre o código b110. É constituído pelo `_id`, que é gerado automaticamente, pelo título do código (`name`), pelo código (`code`), a respetiva descrição (`description`), dois *arrays* com exemplos que este domínio pode incluir ou excluir, e por fim um campo com o histórico de alterações realizadas neste documento (`changes`). Este

último campo é um objeto com dois *arrays* de documentos, `inclui` e `exclui`. Caso ocorra uma alteração no campo `inclui`, é guardado no objeto `changes`, dentro do *array* `inclui`, um documento novo com o nome do utilizador que efetuou a alteração (`user`), a data e hora (`time`), e a respetiva alteração (`descpt`).

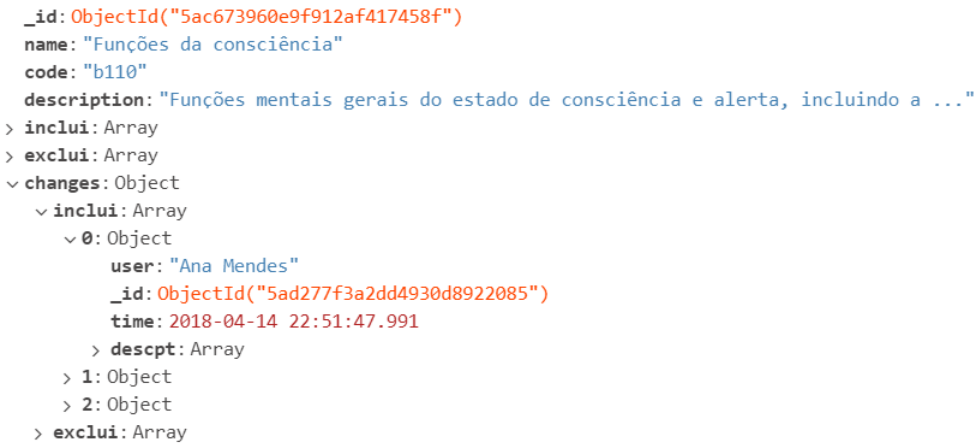

Figura 28 - Exemplo da estrutura de um documento.

4.3.2 Servidor

Relativamente ao resto do projeto, a aplicação encontra-se na pasta `PCCIF`. Sendo que o servidor está localizado na raiz dessa pasta e o cliente dentro da designada `Client`. Para executar o servidor é necessário introduzir na linha de comandos a instrução `nodemon app.js`, dentro do diretório onde este se encontra. Na Figura 29 é possível observar a estrutura da pasta. Através da execução de comandos como `npm init` e `ng new` são criados diversos ficheiros de configuração e no caso do último, para além destes ficheiros, ainda são inicializadas várias bibliotecas e o componente base da aplicação. Neste documento apenas irão ser explicados os ficheiros que sofreram alterações ou que foram criados de raiz.

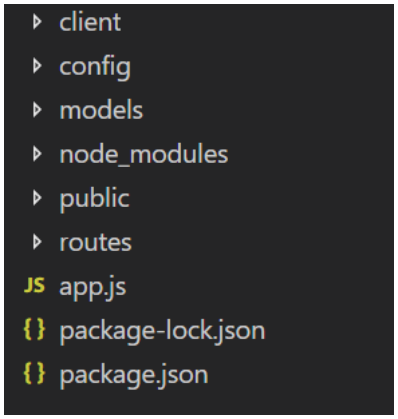

Figura 29 - Estrutura do projeto da aplicação.

Package.json

Contém toda a informação importante sobre a aplicação e as dependências necessárias para construir e executar a aplicação, como por exemplo o nome da aplicação, das dependências e as suas versões.

App.js

Este é o ficheiro principal do servidor Express, onde se encontram os *imports* de todas as dependências e a ligação à base dados. São criadas instâncias para importar o Express dos módulos (node_modules), o Path que auxilia no caminho dos ficheiros, o Body Parser que possibilita o *parse* do JSON, o Cross-Origin Resource Sharing (CORS) e por fim, o Mongoose que é conectado ao MongoDB na porta 27017. O servidor é iniciado neste ficheiro, na porta 3000.

Cors.js

Neste ficheiro encontra-se a configuração do CORS, que é um mecanismo que permite que recursos restritos de uma página sejam solicitados por um domínio fora daquele de onde o recurso teve origem. Isto é, permite que outros domínios acedam aos recursos do *back-end* que necessitem de ser expostos, assim como chamadas à API REST [92].

Na aplicação em questão, foi utilizado o Angular-cli[79], que possibilita um servidor *standalone* (que executa sozinho não fazendo parte de nenhum grupo) para as aplicações em Angular. Ou seja, o *front-end* e o *back-end* são executados em domínios distintos, com portas diferentes. Enquanto o domínio do servidor é http://localhost:3000, o do cliente é http://localhost:4200.

Route.js

Neste ficheiro estão definidas todas as funções da REST API. Contém um conjunto de *router handlers* definidos para lidar com os pedidos HTTP provenientes do cliente. Os *router handlers* definem a estrutura do URL que o cliente utiliza para interagir com a aplicação *web* e utilizam diversos métodos para definir as rotas como *router.all(), router.get(), router.post(), router.delete()* e *router.put().* A rota é utilizada para lidar com os pedidos HTTP e é uma combinação entre um caminho e uma função *callback*, que é executada quando o caminho do pedido corresponde ao caminho da rota. A função *callback* é denominada de *router handler*. Cada método utilizado gere tipos de pedidos HTTP diferentes. Por exemplo, o *router.get()* apenas lida com pedidos GET, o *router.post()* com os pedidos POST, o *router.all()* lida com todos os tipos de pedidos e assim sucessivamente.

Cada *router handler* recebe uma referência para os objetos do pedido (`req`) e da resposta (`res`) do pedido HTTP que está atualmente a ser atendido. Podem ser executados múltiplos *router handlers* para um único pedido HTTP e deve terminar o pedido ou executar a função *next().* Este método é utilizado para invocar o próximo *route handler* que corresponde ao caminho da rota.

A descrição de todos os *router handlers* definidos neste ficheiro encontram-se descritos detalhadamente no Anexo A.

[79] https://cli.angular.io/

Models

Na pasta Models encontram-se cinco ficheiros com *schemas* definidos e modelos exportados. Os *schemas* mapeiam uma coleção do MongoDB e definem o formato dos documentos existentes nesta. Para utilizar este *schema* é necessário convertê-lo num modelo. As instâncias dos modelos são documentos, que dispõem de uma série de métodos incluídos (CREATE, SAVE, REMOVE e FIND) que permitem criar, ler, alterar e remover instâncias da base de dados. A descrição de cada ficheiro encontra-se na Tabela 15.

Tabela 15 - Descrição dos ficheiros que possuem os schemas do Mongoose.

Ficheiros	Descrição
codes.js	Definição do *schema* relativo aos documentos detentores dos dados da CIF, contidos na coleção "data".
forumPosts.js	Definição do *schema* relativo aos documentos detentores dos dados dos tópicos do fórum, contidos na coleção "data".
proposals.js	Definição do *schema* relativo aos documentos detentores das propostas realizadas pelos utilizadores sobre a CIF, contidos na coleção "data".
user.js	Definição do *schema* relativo aos documentos detentores dos dados dos utilizadores registados, contidos na coleção "users".
userstoapprove	Definição do *schema* relativo aos documentos detentores dos dados dos utilizadores não registados, contidos na coleção "data".

No ficheiro `user.js`, para além da definição do *schema* dos documentos dos utilizadores registados, também estão algumas funções importantes (`addUser()` e `comparePassword()`) relativas ao *hashing* e à comparação da *password* inserida pelo utilizador. Para tal foram utilizados os pacotes Passport[80] e Bcrypt[81] como *middleware* para realizar a autenticação do utilizador. As definições de alguns conceitos importantes para a compreensão deste processo encontram-se descritos na Tabela 16.

Tabela 16 - Descrição de conceitos e ferramentas importantes para o login e o registo [93] [94] [95].

Conceitos	Descrição
Hashing	Ato de converter *passwords* em *Strings* ilegíveis de carateres que são desenhados para serem impossíveis de converter, denominados de *hashes*.
Passport	É um *middleware* de autenticação para o Node.js, utilizado para autenticar pedidos. Fornece uma série de estratégias diferentes e fáceis de implementar, permitindo assim integrar técnicas de *login* (verificação do *username* e da *password*) para os vários tipos de serviços.
Bcrypt	Biblioteca do NPM que contém funções que facilitam o *hashing* e a comparação de *passwords* no Node.js. Utilizada para complementar o Passport.
Salting	Para evitar a pré-computação, é utilizado o truque denominado *Salting*, onde são adicionados dados aleatórios à palavra-passe antes de realizar o *hashing* Esta combinação chama-se *salt*.

[80] http://www.passportjs.org/
[81] https://www.npmjs.com/package/bcrypt

A estratégia do Passport utilizada foi a "passport-jwt" que utiliza o padrão JSON Web Token[82] (JWT), que funciona através da atribuição e transmissão de *tokens* encriptados em pedidos, que ajudam a identificar o utilizador conectado, em vez de armazenar o mesmo numa sessão no servidor e criar uma *cookie*. Os *tokens* são enviados no cabeçalho da autorização do método HTTP, de onde o Passport os irá extrair e validar e possuem uma data de validade que neste caso é uma semana, ou seja, ao fim deste tempo o utilizador necessita de realizar novamente o *login* na aplicação.

A função `addUser()` é invocada quando o utilizador se regista na aplicação, recebendo toda a sua informação. Aqui irá ser gerado o *salt*, e de seguida, este é utilizado para realizar o *hashing* através das função `genSalt()` e `hash()` da biblioteca Bcrypt. No fim é guardado o *hash* em vez da password em *plain text*.

A função `comparePassword()` é invocada quando o utilizador efetua o *login* na aplicação e recebe como parâmetros a palavra-passe que este inseriu e o *hash* que está guardado na base de dados, correspondente ao seu *username*. Dentro desta é utilizada a função `compare()` da biblioteca Bcrypt, que irá retirar o *salt* do *hash* e comparar este à password inserida pelo utilizador. Se for igual então é criado um *token* encriptado com o JWT na Restful API que identifica este utilizador, mantendo o seu *login* por uma semana.

Config

Esta pasta dispõe do ficheiro referente à configuração da autenticação para proteger recursos e restringi-los apenas a utilizadores verificados. Aqui é aplicada a estratégia *passport-jwt*[83] do módulo Passport, onde é verificado se o *token* enviado pelo utilizador no pedido é válido ou não foi manipulado. O resultado desta validação irá decidir se o utilizador tem acesso à rota `/users/profile` por exemplo.

4.3.3 Cliente

O lado do Cliente encontra-se na pasta `Client` do projeto e para ser iniciado é necessário introduzir na linha de comandos a instrução `ng serve`, dentro desse mesmo diretório. A aplicação pode ser acedida através do domínio `http://localhost:4200`. A estrutura da pasta que detém o lado do cliente encontra-se disposta na Figura 30. Assim como na secção 4.3.2, apenas serão descritos os ficheiros que foram criados/alterados.

[82] https://jwt.io/
[83] http://www.passportjs.org/packages/passport-jwt/

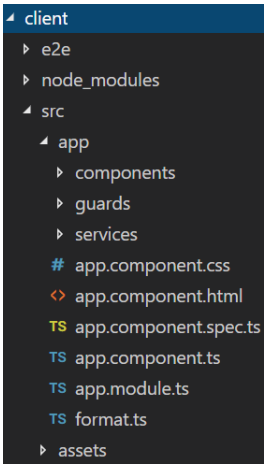

Figura 30 - Estrutura do ficheiro do lado do cliente.

App.module

Este ficheiro é um mecanismo para agrupar componentes, diretivas e serviços relacionados, de forma a que possam ser combinados com outros módulos para criar a aplicação. Deste modo, todos os componentes e serviços devem ser importados neste ficheiro e declarados para poderem ser utilizados por outros módulos. Possui um *decorator* `@NgModule` que identifica o AppModule como uma classe "módulo" do Angular e utiliza um objeto de metadados que indica ao Angular como compilar e iniciar a aplicação [96]. Este *decorator* possui [96]:

- *Imports* – Contém todas as bibliotecas necessárias à execução da aplicação. Inclui por exemplo o `BrowserModule` que todas as aplicações necessitam para correrem no *browser*;
- Declarações – Dispõe de todos os componentes presentes na aplicação;
- *Bootstrap* – Componente raiz que o Angular cria e insere na página `index.html`;
- *Providers* – Possui os Serviços todos de forma a estarem disponíveis para toda a aplicação.

Além do *decorator*, inclui também um *array* do tipo Routes, onde cada objeto consiste em duas propriedades: caminho e o componente. Neste objeto são declarados todos os URLs da aplicação e "ligados" a um componente. Ou seja, quando um URL é acedido, será invocado o componente que lhe corresponde, declarado previamente neste *array* e o seu conteúdo é disposto no local onde se encontra o elemento `<router-outlet>` no ficheiro `app.component.html`. Para ativar esta configuração, é necessário adicionar o *array* à propriedade *imports* do decorator @NgModule.

Componentes

Um componente é o bloco de construção principal de uma aplicação em Angular 2 e é considerado uma visão particular da aplicação que contém os seus próprios dados e lógica [72]. Podem existir inúmeros componentes numa aplicação e cada um é constituído por quatro ficheiros:

- Um ficheiro CSS para os estilos;
- Um ficheiro HTML para o *template*;
- Um ficheiro TypeScript com a classe do componente;
- Um ficheiro teste para a classe (ficheiro anterior).

No projeto atual existem dezanove componentes e a descrição do propósito de cada um é apresentada na tabela do Anexo C. Os únicos ficheiros alterados neste projeto foram o HTML e o TypeSript. A descrição detalhada de cada uma das funções presentes neste último ficheiro encontram-se no Anexo A.

Serviços

Este projeto é constituído por seis Serviços, onde no Anexo D é possível averiguar a descrição detalhada de cada função integrante dos mesmos. Os Serviços são utilizados pelos Componentes para enviar e receber dados. Funcionam como um *wrapper* do lado do cliente para os *endpoints* da RESTful API. Os Componentes "consomem" os Serviços, ou seja, é possível injetar um Serviço num Componente, concedendo a este acesso à classe do Serviço. Dentro do Serviço, através do *import* do protocolo HTTP, é possível realizar pedidos ao servidor. Utilizando a instância *http* da classe, é possível invocar os métodos GET, PUT, POST ou DELETE, passando um URL que será o *endpoint* e possíveis parâmetros em formato JSON.

Guardas

As Guardas de rotas são *interface*s que informam o *router* (permite a navegação de uma *view* para outra conforme os utilizadores executam ações [97]) do Angular se este deve permitir a navegação para uma determinada rota solicitada pelo utilizador ou não. Esta decisão é tomada através de um valor de retorno verdadeiro ou falso de uma classe que implementa a *interface* da Guarda. O comportamento do *router* é diferente consoante o Guarda que é utilizado e existem cinco tipos diferentes [97]:

- `CanActivate` – Decide se a rota pode ser ativada;
- `CanActivateChild` – Decide se as rotas filhas de uma rota podem ser ativadas;
- `CanDeactivate` – Decide se a rota pode ser desativada;
- `CanLoad` – Decide se o módulo pode ser carregado;
- `Resolve` – Apenas ativa a rota após os dados se encontrarem disponíveis.

Na aplicação foram utilizadas cinco Guardas de rota, todas do tipo `CanActivate` e a descrição de cada uma encontra-se presente no Anexo E.

4.3.4 MVC

A aplicação não comunica diretamente com o servidor, utilizando a REST API para receber e enviar dados. Através das rotas criadas no Angular 2 é possível navegar pelas páginas *web*, porém todos os dados são exibidos através da atualização de uma única página (SPA).

De acordo com a arquitetura MVC, os *models*, *views* e *controllers* encontram-se devidamente separados, uma vez que os *models* são implementados no lado do servidor, enquanto que as *views* e os *controllers* são implementados no lado do cliente. Traduzindo esta arquitetura para o projeto referido (Tabela 17), os *models* são os modelos criados através da *framework* Mongoose, o conjunto dos componentes com as suas respetivas páginas HTML formam uma *view* e por fim os *controllers* são os Serviços que são consumidos pelos Componentes. Quando um pedido é feito pelo cliente é enviado para o *controller*. Este se necessário realiza um pedido ao modelo e o modelo responde ao *controller*. Este por sua vez envia a resposta para a *view* e esta renderiza a página HTML para o *browser* do cliente.

Tabela 17 - MVC na Mean Stack.

MVC	MEAN
Model	Modelo Mongoose
View	Componentes + HTML
Controller	Serviços

CAPÍTULO 5

Protótipo da Aplicação *Web*

Neste capítulo é apresentado o protótipo da aplicação *web* desenvolvida. Pode-se encontrar uma série de imagens das páginas mais relevantes, com a descrição detalhada das suas funcionalidades e ainda uma avaliação das suas características gerais.

5.1 Descrição

PCCIF é uma aplicação *web* com o intuito de disponibilizar uma *interface user friendly* para a codificação de intervenções clínicas e respetivos resultados. Dispõe de um *design* simples, intuitivo e consistente, essencialmente com cores neutras. Oferece as seguintes funcionalidades:

- Consulta da classificação CIF com uma pesquisa personalizada;
- A submissão de propostas de exemplos para as instâncias inclui e exclui dos códigos, de modo a enriquecer a classificação;
- Um fórum para discutir assuntos relacionados com a CIF.

É indicada principalmente para pessoas que trabalhem na área da saúde, mas também pode ser utilizada na área da educação e investigação. Para aceder às duas últimas funcionalidades é necessário efetuar o registo na aplicação, o qual será avaliado pelo administrador, uma vez que apenas indivíduos com conhecimento e experiência na CIF podem submeter dados. A avaliação das propostas submetidas também é realizada pelo administrador profissional nesta área, assegurando a validação dos dados inseridos na CIF.

5.1.1 *Interface*

A *interface* da plataforma é consideravelmente simples antes de efetuar o *login*, uma vez que a maior parte das suas funcionalidades e vantagens apenas são disponibilizadas após o início da sessão. Para um utilizador sem o *login* efetuado a única utilidade de que dispõe é a consulta da CIF e um método de pesquisa personalizado sobre a mesma. Para um utilizador com o registo aprovado, para além desta funcionalidade, pode ainda sugerir a introdução de um exemplo no campo inclui ou exclui, que pode ou não ser aprovado pelo administrador. Ainda pode usufruir de um fórum para discutir possíveis exemplos a submeter na CIF ou outros temas relacionados com a mesma, e uma página com todas as propostas submetidas pelos utilizadores, com o seu estado e o direito a votar sobre as mesmas que ainda não foram aprovadas pelo administrador.
Por outro lado, o administrador pode desfrutar de mais funcionalidades. Para além das mesmas que um utilizador comum possui, pode adicionar ou editar diretamente os campos inclui e exclui da CIF, e ainda apagar tópicos do fórum. Por fim, possui ainda uma página exclusiva, com um menu lateral com múltiplas opções para a gestão dos utilizadores, das suas sugestões, dos tópicos do Fórum e para visualização de todo o histórico de alterações. De seguida serão explicadas todas estas funcionalidades ao pormenor

5.1.2 Menu

A Figura 31 apresenta a menu antes do utilizador efetuar o registo ou *login*. Apenas dispõe o acesso à página Códigos, onde se encontra a classificação CIF para consulta e pesquisa. Na página do registo todos os campos são obrigatórios exceto o número de ordem ou o comentário. Caso o utilizador possua um documento de uma entidade emissora relacionada com a saúde, indicando assim que é um profissional nesta área, é obrigado a introduzir o número do seu documento e a respetiva entidade emissora, não necessitando de preencher o campo do comentário. Caso não encaixe neste tipo de utilizador, e seja por exemplo um professor, aluno ou investigador, necessita de preencher o campo do comentário, descrevendo o porquê de pretender uma conta na plataforma. Ambos os casos serão verificados pelo administrador, que decide se o registo é válido ou não.

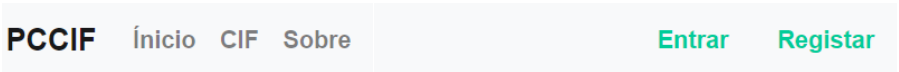

Figura 31 - Menu antes de efetuar login.

Após o utilizador efetuar o *login*, dispõe da mesma página dos códigos, porém já com a opção de adicionar exemplos, nos campos inclui e exclui da CIF. O Fórum também é uma página exclusiva de utilizadores com *login*, assim como o Perfil e a página das Propostas (Figura 32).

Figura 32 - Menu após o login efetuado.

5.1.3 CIF

A Figura 33 apresenta a página dos Códigos, onde se encontra a documentação da CIF. Como é possível observar, dispõe de um campo de pesquisa, com opção de colocar filtros (nome, código, descrição, inclui, exclui), caso o utilizador pretenda pesquisar uma palavra apenas num determinado campo, ou em vários simultaneamente. Relativamente à tabela da CIF, cada linha é clicável e demonstra o nível seguinte do código.

Figura 33 - Página dos Códigos da CIF.

A partir do nível três dos códigos, aparece um botão que possibilita ao utilizador adicionar um novo exemplo no campo inclui ou exclui. Se for administrador, aparece um botão adicional, o Editar, que permite ao administrador alterar ou apagar um exemplo já existente nos campos inclui e exclui.

Figura 34 - Adicionar/Editar campos na CIF.

Quando o utilizador clica no botão Adicionar observado na Figura 34, é redirecionado para a página demonstrada na Figura 35. Esta apresenta o código que foi selecionado para adicionar informação e os campos que são necessários preencher para prosseguir. Estes são o campo da CIF (inclui ou exclui) e a descrição do exemplo que pretende adicionar. Após a submissão, necessita de esperar pela aprovação do administrador. Caso seja este último a realizar a submissão, é adicionada automaticamente.

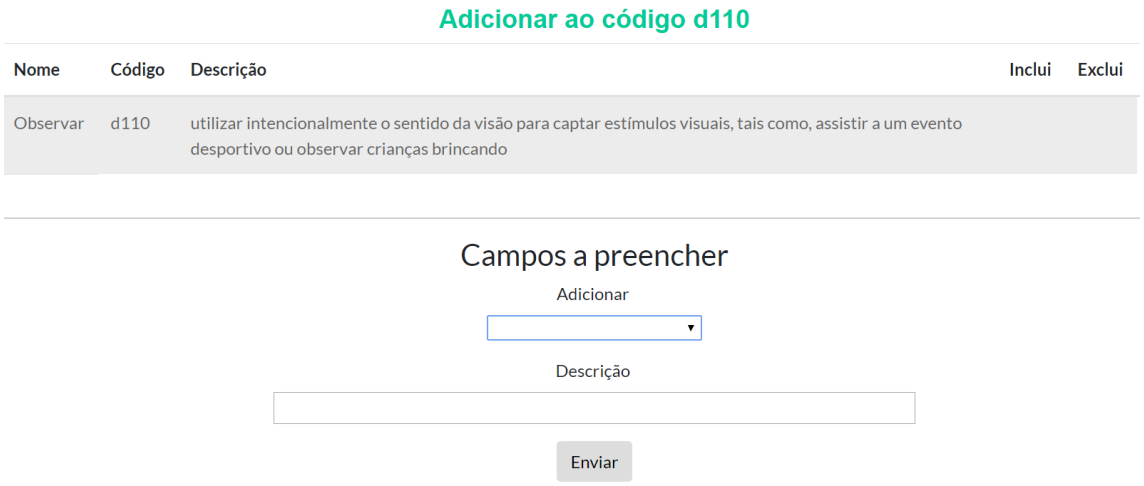

Figura 35 - Adicionar inclui/exclui na CIF.

5.1.4 Fórum

Na Figura 36 é possível observar a página do Fórum, onde estão dispostos os tópicos aprovados pelo administrador. Estes estão divididos em duas secções: tópicos em discussão (é possível comentar) e os tópicos concluídos (não é possível comentar). Oferece opções como criar um novo tópico, pesquisar uma discussão e caso seja o administrador permite apagar a publicação (*icon* do lixo do lado direito) ou conclui-la (icon com um certo do lado direito), sendo que esta última opção apenas é possível quando se encontra na secção dos tópicos em discussão.

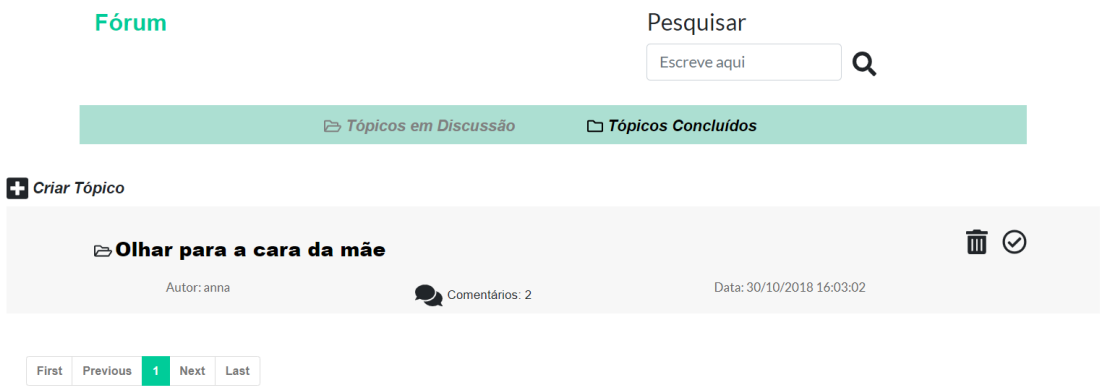

Figura 36 - Página do Fórum.

Ao carregar num tópico, o utilizador é redirecionado para uma nova página com os detalhes do mesmo, onde pode introduzir o seu comentário, caso não seja um tópico concluído (Figura 37).

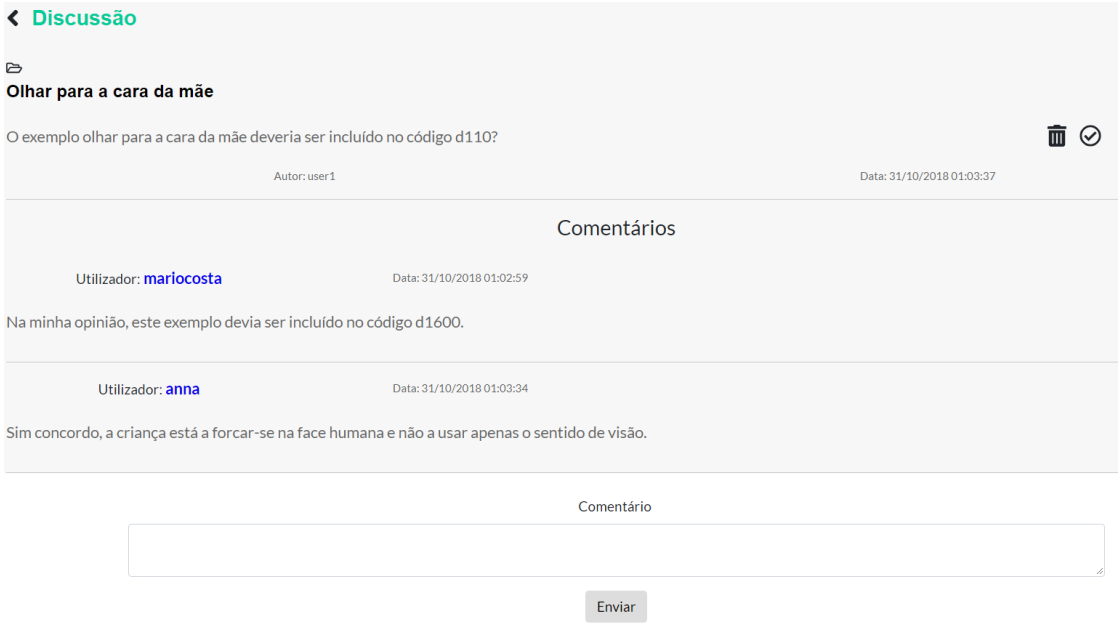

Figura 37 - Página dos comentários de um tópico do Fórum.

Caso o utilizador selecione o botão de adicionar um tópico no fórum, é redirecionado para uma nova página onde necessita de preencher dois campos obrigatórios: título e descrição.

5.1.5 Propostas

Esta página proporciona a visualização de todas as propostas submetidas que se encontram aprovadas, por aprovar e rejeitadas. Com todas as suas informações e ainda os nomes dos utilizadores que as submeterem. Na secção das propostas por aprovar é possível votar de forma positiva ou negativa, de modo a facilitar a sua aprovação, uma vez que se encontra ordenada de forma decrescente de votos (Figura 38). Além disto, permite ainda a pesquisa sobre as mesmas

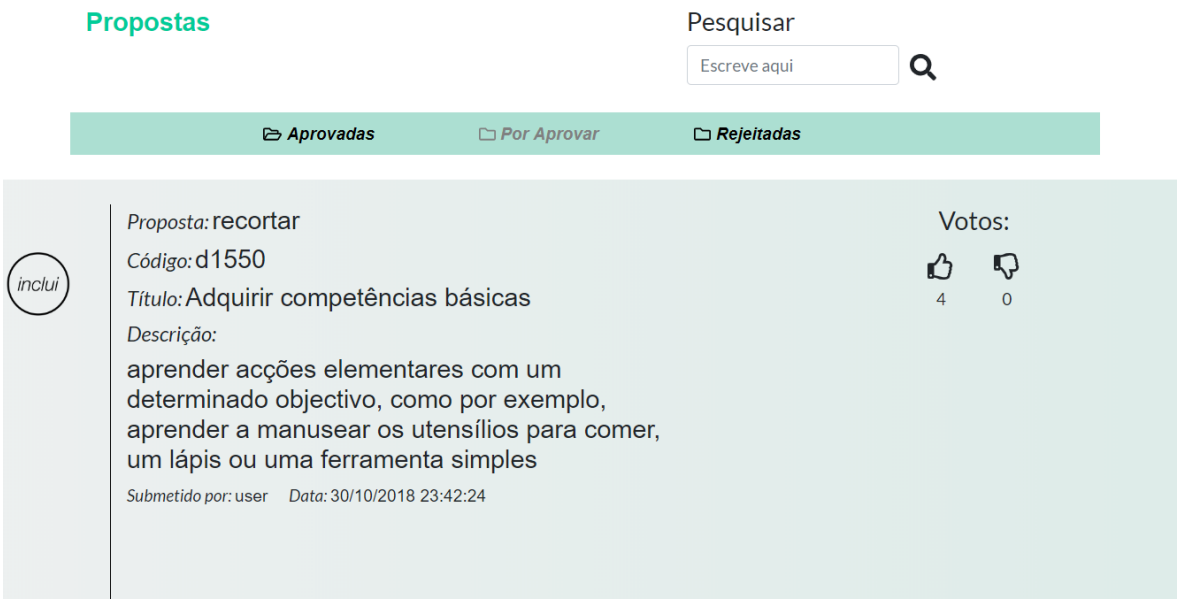

Figura 38 - Página das Propostas.

5.1.6 Perfil

A última página disponível a todos os utilizadores registados é o Perfil (Figura 39). Nesta é possível verificar as suas informações pessoais, editá-las (lado esquerdo da figura) e ainda examinar todas as suas propostas submetidas e qual o estado em que se encontram (lado direito da figura).

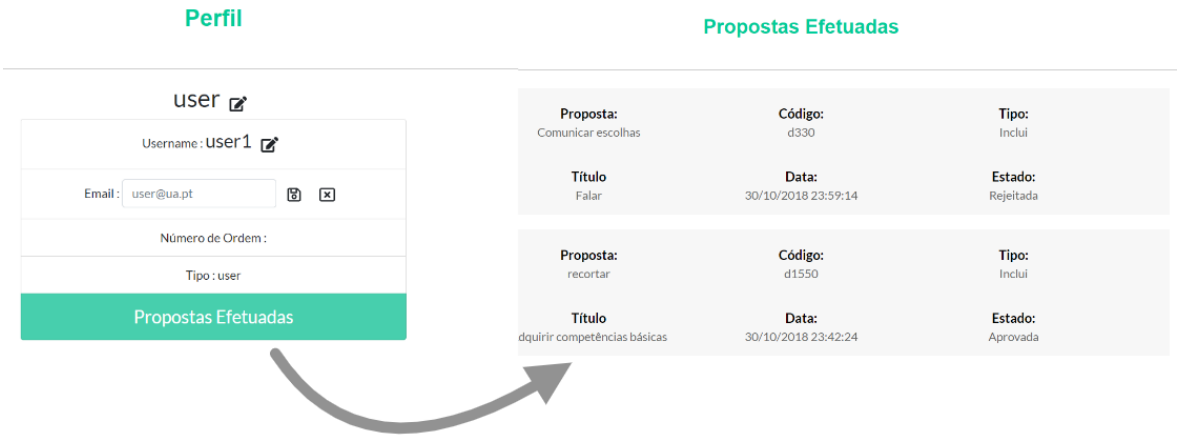

Figura 39 - Página do Perfil e das Propostas efetuadas.

5.1.7 Administrador

Como já foi referido anteriormente, o administrador para além das outras opções que os utilizadores regulares possuem, dispõe de uma página exclusiva: Gerir (Figura 40). Os detalhes desta serão descritos ao longo desta secção.

Figura 40 - Menu do administrador.

A página Gerir faculta diversas opções ao administrador, que se encontram no lado esquerdo da Figura 41. A primeira opção é relativa às sugestões que os utilizadores propuseram sobre os campos inclui ou exclui, e é demonstrada na figura mencionada. Ao carregar numa das sugestões, é apresentada uma nova página com os seus detalhes.

Figura 41 - Menu lateral do administrador e sugestões dos utilizadores.

A Figura 42 revela a página de detalhes das sugestões dos utilizadores, como foi referido anteriormente. O administrador conta com dois botões que aprovam ou rejeitam a sugestão do utilizador.

Figura 42 - Validação das sugestões dos utilizadores.

A segunda funcionalidade no menu lateral do administrador é aprovação dos registos realizados pelos utilizadores. Neste caso, conta novamente com dois botões, aprovar ou rejeitar o registo (Figura 43).

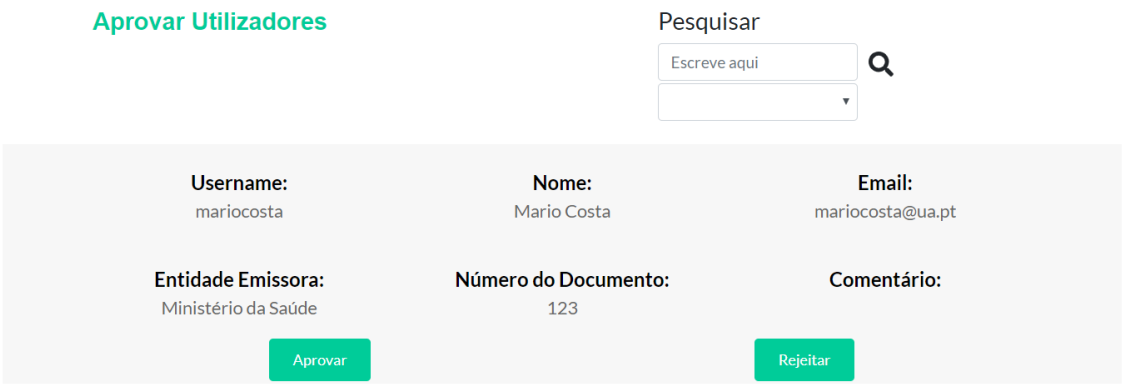

Figura 43 - Validação dos registos dos utilizadores.

Outra opção da barra de navegação lateral é a pesquisa de utilizadores já registados. Possui ainda a possibilidade de pesquisar com filtros (*username*, nome, *email* ou número de ordem).

A quarta funcionalidade é o histórico, onde estão dispostas todas as alterações realizadas sobre os campos inclui e exclui de um determinado código (Figura 44). Dispõe também a opção de pesquisa com diversos filtros.

Figura 44 - Histórico das alterações realizadas.

Por fim, encontra-se a validação das publicações criadas no fórum efetuadas pelos utilizadores. Contém as opções de aprovar ou rejeitar o tópico, como se pode observar na Figura 45. Caso seja aprovado, passa a ser visível na página do Fórum para todos os utilizadores poderem comentar.

Figura 45 - Validação de publicações no Fórum.

5.2 Características gerais

5.2.1 Validação dos dados

Foram implementadas no sistema funções que validam os dados a serem armazenados, de modo a garantir a sua integridade e compatibilidade. A validação dos dados é realizada depois do utilizador fazer o pedido de armazenamento de dados. Estes são validades por determinadas funções e, no caso da existência de erros é apresentada uma mensagem com cor vermelha indicando a causa do erro. Alguns exemplos possíveis são:

- O *email* inserido não corresponde ao formato convencional na página de registo.
- Quando o utilizador insere um nome de utilizador ou *email* que já existem na base de dados quando procede ao registo.
- Caso não preencha todos os campos obrigatórios na página de *login* ou do registo.
- A palavra-passe inserida no *login* não corresponde à guardada na base de dados.

5.2.2 Processamento dos dados

Para que o utilizador obtenha um *feedback* constante do sistema, foram implementados avisos de forma a informar o utilizador do estado das operações por ele requeridas. Por exemplo, quando são submetidos dados e realizadas pesquisas é apresentada uma mensagem indicando se a operação foi bem sucedida ou não. No caso da submissão de dados é possível observar uma faixa com a mensagem e uma cor de fundo personalizada (verde para sucesso e vermelho para insucesso). Relativamente à pesquisa, quando não são encontrados resultados que correspondem ao que é pretendido, é exibida uma mensagem indicando que não foram encontrados resultados, no local onde estes deveriam estar dispostos.

5.2.3 Prevenção de erro

Existem dois tipos de erros que os utilizadores normalmente cometem: o deslize e o engano. O deslize é quando um utilizador pretende realizar uma ação, mas acaba por efetuar outra, que ocorre geralmente quando este não se encontra totalmente focado. Já o engano é quando a compreensão de alguma informação é equivocada ou entendida de outra forma [98]. Assim sendo, a aplicação dispõe de caixas de confirmação sempre que ocorre a submissão de qualquer tipo de dados, com o propósito de prevenir este tipo de erros.

5.2.4 Estética

Quanto maior a quantidade de informação, maior será a quantidade de dados que serão analisados e decisões que o utilizador precisará de tomar. Desta forma, é crucial manter apenas o que é realmente relevante e desta forma as informações secundárias podem ser deixadas em segundo plano. A aplicação apresenta um *design* bastante minimalista, com pouco texto e botões intuitivos, tornando-a assim mais eficiente na transmissão de informação aos utilizadores.

5.2.5 Consistência

Manter consistência entre as páginas de uma aplicação é essencial para que não seja necessário o entendimento de vários padrões e formas de interações distintas. Além disto, a experiência da utilização torna-se muito mais interessante pois não existe uma sensação de estar perdido. Muitas vezes o motivo pelo qual os utilizadores não interagem com as aplicações é devido a essa sensação causada pela falta de consistência e padronização. No caso da aplicação *web* em questão, todas as páginas seguem o mesmo padrão, com menus e cores constantes, facilitando assim a interação e a aprendizagem.

CAPÍTULO 6

Conclusão e Trabalho Futuro

A CIF é uma classificação que representa um marco conceptual para descrever a saúde e os estados relacionados com a mesma, constituindo um valioso instrumento de utilidade prática na Saúde Pública. Apesar da importância e atualidade da CIF, alguns conceitos desta foram pouco detalhados e justificados, podendo ocasionar interpretações distintas. É aqui que surge a necessidade de precisar a informação de forma a que haja um aumento de detalhe, e desta forma a codificação torna-se menos suscetível a erros, mais simplista e mais rápida para aqueles que queiram fazer uso da CIF. O facto da codificação da CIF ser feita manualmente até ao momento, teve impacto na evolução da mesma, uma vez que apenas a partir da prática e utilização se consegue detetar erros e constatar melhorias [7]. Além disto, a catalogação realizada por um profissional não é usufruída pelos outros, uma vez que não têm acesso, podendo até ser perdida ou mesmo esquecida.

Face a este problema, surge o principal objetivo deste trabalho, o qual se prende com a criação de uma aplicação *web* que facilite e armazene o processo de catalogação em intervenções clínicas essencialmente na área da Terapia Ocupacional, o que é possível atingir visto que existem inúmeras tecnologias que permitem alcançar este propósito. Desta forma, optou-se pela utilização da MEAN *Stack*, que é um conjunto de tecnologias que permitem a criação de uma plataforma *online* que suporta uma grande quantidade de dados.

Assim, foi criada uma nova plataforma, simples, interativa e a qual pretende inovar o que até hoje em dia existe. Enquanto as plataformas semelhantes existentes permitem apenas uma consulta ou introdução de novos ramos na “árvore” da CIF, este projeto prende-se com uma complementação dos ramos já existentes, e desta forma facilitar e auxiliar os utilizadores na sua interação com a CIF. Para permitir uma experiência mais completa foram adicionadas ainda diversas funcionalidades que não estavam previstas inicialmente, nomeadamente um fórum de discussão, a consulta e votação de propostas submetidas por outros utilizadores e consulta de um histórico de submissões e alterações.

Esta aplicação procura revolucionar o processo de codificação principalmente na área da Fisioterapia Ocupacional. Com o armazenamento de catalogações realizadas por

profissionais numa plataforma unificada e partilhada, minimizado assim o risco de perda ou esquecimento de dados. Além disto, através da simplificação da catalogação de intervenções clínicas, apresenta-se como uma vantagem para indivíduos com pouca experiência, uma vez que ao longo da sua utilização o número de exemplos e o nível de detalhe da CIF vai aumentando.

Por forma a assegurar a qualidade da base dos dados nem todos os utilizadores podem registar-se na aplicação, pois esta pressupõe uma aprovação por parte do administrador-entidade máxima. Só se podem registar utilizadores que possuam uma entidade emissora (p.e. ordem dos médicos ou ministério da saúde) ou então um professor ou investigador que tenha uma razão válida para pretender o acesso a esta plataforma. Desta forma apenas pessoas creditadas poderão submeter exemplos para CIF (que também necessitam de aprovação do administrador), pois assim parte-se do pressuposto que a informação submetida na plataforma seja uma informação que provenha de uma fonte segura. Assim passa a existir a categoria de administrador(es) e utilizadores registados. Para que a informação seja discutida entre peritos, e que através dessa discussão se possam obter/adicionar dados mais fidedignos (com suporte credível), a plataforma permite ainda que os diversos utilizadores possam votar e dar a sua opinião sobre as diversas propostas, novos tópicos, entre outros.

Para um trabalho futuro seria proveitoso desenvolver outras funcionalidades para refinar o processo de aprovação/rejeição de propostas submetidas por utilizadores. Como por exemplo, a possibilidade de o administrador alterar uma sugestão sem necessitar de a rejeitar. Seria também interessante e útil explorar a vertente da inteligência artificial na medida em que, se possa anular o efeito das redundâncias frásicas e introduzir o *auto-complete* em pesquisas. Além disto, seria proveitoso realizar alguns testes sobre a aplicação, como por exemplo testes de carga e de desempenho, simulando um grande número de utilizadores e transações simultâneas.

Referências

[1] H. B. V. Di Nubila and C. M. Buchalla, “A classificação internacional de funcionalidade, incapacidade e saúde da organização mundial da saúde: conceitos, usos e perspectivas,” *Rev. Bras. Epidemiol.*, vol. 8, no. 2, pp. 187–193, 2005.

[2] A. Leitão, “CIF: Classificação Internacional de Funcionalidade,” *Calssificação Int. funcionalidade, incapacidade e saude*, p. 238, 2004.

[3] E. S. de Araújo, “Classificação Internacional de Funcionalidade, Incapacidade e Saúde - (CIF) em Fisioterapia: Uma revisão bibliográfica - Dissertação de Mestrado,” p. 117, 2008.

[4] E. S. de Araújo, “Classificação Internacional de Funcionalidade, Incapacidade e Saúde: prespetivas emergentes para a Fisioterapia.,” 2010.

[5] H. B. V. Di Nubila and C. M. Buchalla, “O papel das Classificações da OMS - CID e CIF nas definições de deficiência e incapacidade,” *Rev. Bras. Epidemiol.*, vol. 11, no. 2, pp. 324–335, 2008.

[6] N. P. Rocha, “Tecnologias da Informação e Deficiência,” Aveiro, 2003.

[7] L. Castaneda, A. Bergmann, and L. Bahia, “The International Classification of Functioning, Disability and Health: a systematic review of observational studies,” *Rev BRas epidemiol*, pp. 437–451, 2014.

[8] M. C. Battisti, “Funcionalidade humana Políticas Públicas Classificação Internacional de Funcionalidade e Incapacidades em Saúde.”

[9] A. I. G. Rochains, “Avaliação da usabilidade de produtos e serviços ‘Ambient Assisted Living’ numa abordagem ‘Living Lab,’” Universidade de Aveiro, 2016.

[10] R. F. Sampaio and M. T. Luz, “Funcionalidade e incapacidade humana: explorando o escopo da classificação internacional da Organização Mundial da Saúde,” *Cad. Saude Publica*, vol. 25, no. 3, pp. 475–483, 2009.

[11] L. Bampi, D. Guilhem, and A. E. Dornelles, “Modelo social : uma nova abordagem para o tema deficiência,” *Rev. Latino-Am. Enferm. Latino-Am. Enferm.*, vol. 18, no. 4, pp. 1–9, 2010.

[12] M. A. De Marco, “Do Modelo Biomédico ao Modelo Biopsicossocial: um projeto de educação permanente,” *Rev. Bras. Educ. Med.*, vol. 30, no. 1, pp. 60–70, 2006.

[13] I. N. para a Reabilitação, “CIF - O que é a CIF? - INR.” [Online]. Available: http://www.inr.pt/content/1/55/que-cif. [Accessed: 26-Oct-2018].

[14] A. Original, “Funcionalidade e incapacidade : aspectos conceptuais , estruturais e de aplicação da Classificação Internacional de Funcionalidade , Incapacidade e Saúde (CIF) Ana Paula Fontes a , *, Ana Alexandre Fernandes b e Maria Amália Botelho c,” vol. 28, no. 2, pp. 171–178, 2010.

[15] M. Riberto, “Core sets da Classificação Internacional de Funcionalidade, Incapacidade e Saúde,” *Rev. Bras. Enferm.*, vol. 64, no. 5, pp. 938–946, 2011.

[16] L. da S. Ferreira, “Medical Information Extraction in European Portuguese,” p. 262, 2011.

[17] O. Organização Mundial da Saúde, “Como usar a CIF: Um Manual Prático para o uso da Classificação Internacional de Funcionalidade, Incapacidade e Saúde (CIF) Versão preliminar para discussão,” p. 106, 2013.

[18] C. Vale, “Classificação Internacional de Funcionalidade (CIF): conceitos , precon ceitos e paradigmas . Contributo de um construto para o percurso real em meio natural de vida .,” *Acta Pediátrica Port. Soc. Port. Pediatr.*, pp. 229–236, 2009.

[19] J. Alvarelhão, A. Queirós, P. Sa-Couto, and N. Rocha, “Goal setting for cerebral palsy children in context therapy: improve reliability when linking to ICF,” *Stud. Health Technol. Inform.*, 2015.

[20] Rehadat, “Rehadat-ICF.” [Online]. Available: https://www.rehadat-icf.de/en/. [Accessed: 27-Sep-2018].

[21] who, “ICF Update Platform,” vol. 0, no. September, 2012.

[22] A. Adhikari, “Full *Stack* JavaScript: Web Application Development with MEAN,” Helsinki Metropolia University of Applied Sciences, 2016.

[23] E. Oliveira, M. J. Varanda, and P. Rangel Henriques, “Compreensão de Aplicações Web: O Processo e as Ferramentas.”

[24] J. Roijackers, “Bridging SQL and NoSQL,” Eindhoven University of Technology, 2012.

[25] A. C. D. Neto, “Bancos de Dados Relacionais: Um guia Completo,” 2011. [Online]. Available: https://www.devmedia.com.br/bancos-de-dados-relacionais/20401. [Accessed: 29-Oct-2018].

[26] A. Martin, “Disadvantages of a Relational Database.” [Online]. Available: https://www.techwalla.com/articles/disadvantages-of-a-relational-database. [Accessed: 29-Oct-2018].

[27] M. Allen, “Relational Databases Are Not Designed For Scale - MarkLogic.” [Online]. Available: https://www.marklogic.com/blog/relational-databases-scale/. [Accessed: 25-Sep-2018].

[28] G. da Cruz Pereira Sousa, “Document-Based Databases: Estudo Comparativo no Âmbito das Bases de Dados NoSql,” Universidade do Minho, 2015.

[29] T. da S. Côrrea, D. E. C. de Almeida, and A. F. G. Neto, “Comparação Entre Banco De Dados Relacional E Nosql,” p. 16, 2015.

[30] F. Customers, “171 Companies that are using MongoDB Database Software.” [Online]. Available: https://www.featuredcustomers.com/vendor/mongodb/customers. [Accessed: 29-Oct-2018].

[31] D. Borowski, “A Guide to Becoming a Full-*Stack* Developer in 2017 – Coderbyte – Medium.” [Online]. Available: https://medium.com/coderbyte/a-guide-to-becoming-a-full-*Stack*-developer-in-2017-5c3c08a1600c. [Accessed: 25-Sep-2018].

[32] J. Koetsier, “Evaluation of JavaScript frame-works for the development of a web-based user interface for Vampires,” 2016.

[33] J. Gordon, “The Pros and Cons of ReactJS for your Online Business.” [Online]. Available: https://www.linkedin.com/pulse/pros-cons-reactjs-your-online-business-jan-gordon.

[Accessed: 30-Oct-2018].

[34] V. Dao, “Development of a front-end application using AngularJS: 1UP Media company case,” University of applied sciences.

[35] A. Alexseyenko, “Angular 2 vs React. What to chose in 2017?,” 2017. [Online]. Available: https://blog.techmagic.co/angular-2-vs-react-what-to-chose-in-2017/. [Accessed: 25-Sep-2018].

[36] Z. Hemel, “Facebook’s React JavaScript User Interfaces Library Receives Mixed Reviews,” 2013. [Online]. Available: https://www.infoq.com/news/2013/06/facebook-react. [Accessed: 09-Nov-2018].

[37] J. Willoughby, “The Top 5 Benefits of React that Make Life Better.” [Online]. Available: https://www.telerik.com/blogs/5-benefits-of-reactjs-to-brighten-a-cloudy-day. [Accessed: 25-Sep-2018].

[38] A. Volski, “Pros and Cons of React Native vs ReactJS.” [Online]. Available: https://torquemag.io/2017/11/pros-cons-react-native-vs-reactjs/. [Accessed: 26-Sep-2018].

[39] K. Simkhada, “Transitioning Angular 2 User Interface (UI) into React Author Title Number of Pages Date Kumar Simkhada Transitioning Angular 2 User Interface (UI) into React,” Helsinki Metropolia University of Applied Sciences, 2017.

[40] Mean.io, “Mongo Express Angular Node.” [Online]. Available: http://mean.io/. [Accessed: 09-Nov-2018].

[41] A. J. Williams, “A comparison of the performance and scalability of relational and document-based web-systems for large scale applications in a rehabilitation context,” University of hertfordshire, 2015.

[42] A. Tiwari, “Top 6 Advantages Of Developing With The MEAN *Stack*.” [Online]. Available: http://www.businesscomputingworld.co.uk/top-6-advantages-of-developing-with-the-mean-*Stack*/. [Accessed: 08-Nov-2018].

[43] Scribd, “Introdução ao Mongoose.” [Online]. Available: https://pt.scribd.com/document/217729752/Nodejs-e-MongoDB-Introducao-ao-Mongoose-NodeBR-NodeJS-Brasil. [Accessed: 08-Nov-2018].

[44] A. R. Gomes de Oliveira and J. D. Zuchi, “MEAN *STACK*: uma solução para o desenvolvimento de aplicações Web.”

[45] Rouse Margaret, “What is MongoDB? - Definition from WhatIs.com.” [Online]. Available: http://searchdatamanagement.techtarget.com/definition/MongoDB. [Accessed: 17-Dec-2017].

[46] A. S. Maurya, “Why MEAN *Stack* for your next WebApp,” 2018. [Online]. Available: https://4thpointer.com/mean-*Stack*-next-webapp/. [Accessed: 13-Nov-2018].

[47] N. Miguel, Q. Arantes, and D. Santos, “Bases de Dados alternativas para Websites.”

[48] MongoDB Docs, “Documents — MongoDB Manual.” [Online]. Available: https://docs.mongodb.com/manual/core/document/. [Accessed: 28-Jul-2018].

[49] A. Nghi and L. Thanh, “Mean *Stack* Web Development,” Centria University of applied

Sciences, 2016.

[50] TutorialsPoint, “MongoDB Overview.” [Online]. Available: https://www.tutorialspoint.com/mongodb/mongodb_overview.htm. [Accessed: 28-Jul-2018].

[51] W3resource, “Databases, documents and collections - w3resource.” [Online]. Available: https://www.w3resource.com/mongodb/databases-documents-collections.php. [Accessed: 28-Jul-2018].

[52] K. Bhamra, “A Comparative Analysis of MongoDB and Cassandra,” University of Bergen, 2017.

[53] M. Docs, “Sharding — MongoDB Manual.” [Online]. Available: https://docs.mongodb.com/manual/sharding/. [Accessed: 08-Nov-2018].

[54] TutorialsPoint, “MongoDB Advantages.” [Online]. Available: https://www.tutorialspoint.com/mongodb/mongodb_advantages.htm. [Accessed: 04-Sep-2018].

[55] DataFlair, “Advantages of MongoDB | Disadvantages of MongoDB,” 2018. [Online]. Available: https://data-flair.training/blogs/advantages-of-mongodb/. [Accessed: 08-Nov-2018].

[56] MongoDB, “GridFS.” [Online]. Available: http://mongodb.github.io/mongo-java-driver/3.2/driver/reference/gridfs/. [Accessed: 15-Nov-2018].

[57] PcMagazine, “Definition from PC Magazine Encyclopedia.” [Online]. Available: https://www.pcmag.com/encyclopedia/term/37486/ad-hoc-query. [Accessed: 17-Nov-2018].

[58] TutorialsPoint, “Node.js Introduction.” [Online]. Available: https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm. [Accessed: 06-Aug-2018].

[59] F. Perdigão De Sousa, “Criação de framework REST/HATEOAS Open Source para desenvolvimento de APIs em Node.js,” Universidade do Porto, 2015.

[60] TeamExtension, “Node.js Portugal.” [Online]. Available: https://teamextension.pt/pt/contratar/dedicado-nodejs-desenvolvedor. [Accessed: 08-Nov-2018].

[61] N. Chrzanowska, “12 Top Applications Written in Node.js - Examples from Big Companies | Netguru Blog on Node.js.” [Online]. Available: https://www.netguru.co/blog/top-companies-used-nodejs-production. [Accessed: 06-Aug-2018].

[62] R. J. Pfitscher, R. R. Obelheiro, and M. A. Pillon, “Sistemas Operacionais - Deadlocks.” [Online]. Available: https://homepages.dcc.ufmg.br/~scampos/cursos/so/aulas/aula9.html. [Accessed: 10-Nov-2018].

[63] Treinaweb, “Node.js: por que você deve conhecer essa tecnologia?” [Online]. Available: https://www.treinaweb.com.br/blog/node-js-por-que-voce-deve-conhecer-essa-tecnologia/. [Accessed: 07-Aug-2018].

[64] TutorialsPoint, “Node.js Express Framework.” [Online]. Available:

https://www.tutorialspoint.com/nodejs/nodejs_express_framework.htm. [Accessed: 07-Aug-2018].

[65] TutorialsPoint, “Express.js.” [Online]. Available: http://www.tutorialsteacher.com/nodejs/expressjs. [Accessed: 17-Aug-2018].

[66] NodeJS Foundation, “How Uber Uses Node.js to Scale Their Business.”

[67] SmartBear, “What is End-to-End Testing.” [Online]. Available: https://smartbear.com/learn/automated-testing/end-to-end-testing/. [Accessed: 18-Nov-2018].

[68] AngularJS, “AngularJS — Superheroic JavaScript MVW Framework.” [Online]. Available: https://angularjs.org/. [Accessed: 19-Dec-2017].

[69] OracleDocs, “What Is a Class?” [Online]. Available: https://docs.oracle.com/javase/tutorial/java/concepts/class.html. [Accessed: 16-Nov-2018].

[70] J. Korva, “Developing a web application with Angular 2.”

[71] AngularIO, “Angular - Architecture overview.” [Online]. Available: https://angular.io/guide/architecture. [Accessed: 14-Nov-2018].

[72] AngularIO, “Angular - Introduction to components.” [Online]. Available: https://angular.io/guide/architecture-components#data-binding. [Accessed: 21-Aug-2018].

[73] AngularIO, “Angular - Introduction to services and dependency injection.” [Online]. Available: https://angular.io/guide/architecture-services. [Accessed: 24-Aug-2018].

[74] P. Le Hégaret, L. Wood, and J. Robie, “What is the Document Object Model? - W3C.” [Online]. Available: https://www.w3.org/TR/DOM-Level-2-Core/introduction.html. [Accessed: 18-Nov-2018].

[75] DevCenter, “Create a Web App and RESTful API Server Using the MEAN *Stack* | Heroku Dev Center.” [Online]. Available: https://devcenter.heroku.com/articles/mean-apps-restful-api. [Accessed: 26-Aug-2018].

[76] J. Cápona, “RESTful API with Node.js,” 2016. [Online]. Available: https://hackernoon.com/restful-api-with-node-js-938c1ae386fe. [Accessed: 25-Aug-2018].

[77] J. F. M. de Peixoto Lima, “Sistema de Monitorização de Vibrações baseado numa arquitetura REST para IoT,” Universidade do Minho, 2016.

[78] R. Moya, “MEAN (Mongo-Express-Angular-Node),” 2014. [Online]. Available: https://jarroba.com/mean-mongo-express-angular-node-ejemplo-de-aplicacion-web-parte-ii/. [Accessed: 26-Aug-2018].

[79] R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” University of California, Irvine, 2000.

[80] M. F. Ribeiro and R.E.Francisco, “Web Services REST Conceitos, análise e implementação,” *Rev. do Inst. Fed. Edução ,Ciência e Tecnol. da Bahia*, no. October 2016, p. 18, 2016.

[81] R. Plansky, “Definição, restrições e benefícios do modelo de arquitetura REST | iMasters.”

[Online]. Available: https://imasters.com.br/desenvolvimento/definicao-restricoes-e-beneficios-modelo-de-arquitetura-rest. [Accessed: 25-Aug-2018].

[82] R. Figueiredo, “Serviço de nomes para acesso de servidores Web a serviços pessoais,” Universidade de Aveiro, 2015.

[83] R. T. Fielding, R. Frystyk, T. Berners-Lee, J. Gettys, and J. C. Mongul, “Hypertext Transfer Protocol-HTTP/1.1,” 1996.

[84] R. T. Fielding and J. Reschke, “Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing,” 2014.

[85] TutorialsPoint, “HTTP Overview.” [Online]. Available: https://www.tutorialspoint.com/http/http_overview.htm. [Accessed: 30-Aug-2018].

[86] DeveloperMozilla, “An overview of HTTP - HTTP | MDN.” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview. [Accessed: 30-Aug-2018].

[87] SumanTechSolutions, “Java Tutorials.” [Online]. Available: http://www.sumantechsolutions.com/java/servlet1.php. [Accessed: 13-Nov-2018].

[88] Netbeans, “A Brief History of NetBeans.” [Online]. Available: https://netbeans.org/about/history.html. [Accessed: 08-Nov-2018].

[89] Netbeans, “NetBeans IDE - Overview.” [Online]. Available: https://netbeans.org/features/. [Accessed: 04-Sep-2018].

[90] V. Le, “An overview of Visual Studio Code for front-end developers.” [Online]. Available: https://medium.freecodecamp.org/an-overview-of-visual-studio-code-for-front-end-developers-49a4aa0771fb. [Accessed: 07-Sep-2018].

[91] S. Basu, “10 Awesome Features of Visual Studio Code -.” [Online]. Available: https://developer.telerik.com/featured/10-awesome-features-of-visual-studio-code/. [Accessed: 07-Sep-2018].

[92] A. Ayaz, “How to handle CORS in an Angular2 and Node/Express Applications.” [Online]. Available: https://medium.com/@ahsan.ayaz/how-to-handle-cors-in-an-angular2-and-node-express-applications-eb3de412abef. [Accessed: 18-Nov-2018].

[93] C. Laviska, “Hashing Passwords with Node.js and Bcrypt · A Beautiful Site.” [Online]. Available: https://www.abeautifulsite.net/hashing-passwords-with-nodejs-and-bcrypt. [Accessed: 19-Nov-2018].

[94] A. Greenberg, “Hacker Lexicon: What Is Password Hashing?,” 2016. [Online]. Available: https://www.wired.com/2016/06/hacker-lexicon-password-hashing/. [Accessed: 19-Nov-2018].

[95] J. Hanson, “Passport GitHub.” [Online]. Available: https://github.com/jaredhanson/passport.

[96] AngularIO, “Angular - NgModules.” [Online]. Available: https://angular.io/guide/ngmodules. [Accessed: 19-Nov-2018].

[97] AngularIO, “Angular - Routing & Navigation.” [Online]. Available: https://angular.io/guide/router. [Accessed: 13-Nov-2018].

[98] A. Freitas, “5 dicas rápidas para um design de interface livre de erros,” 2017. [Online]. Available: https://www.oxigenweb.com.br/artigos/5-dicas-rapidas-para-um-design-de-interface-livre-de-erros/. [Accessed: 16-Nov-2018].

Anexo A - Descrição das rotas da RESTful API.

Na tabela seguinte, estão descritas as rotas da RESTful API que se encontram no ficheiro `router.js`.

Tabela 18 - Descrição dos recursos da API.

URL	Método	Descrição
/base	GET	Procura todos os documentos que possuem um campo *code*, que tenha tamanho um. Retornando assim os códigos *b,s,e* e *d*.
/lvl/:code	GET	Recebe um código e retorna o nível a seguir a esse. Se o código recebido for de tamanho dois então o nível seguinte é constituído por um intervalo de códigos (i.e. b110-b139). Caso contrário, retorna apenas o código seguinte.
/title/:interval	GET	Recebe um intervalo de códigos (i.e. b110-b139) e retorna os códigos que se encontram dentro desse intervalo.
/allUsers	GET	Retorna todos os utilizados da base de dados.
/oneUser/:word/:filter	GET	Recebe uma palavra e um filtro da pesquisa (i.e. nome, *email* ou *username*) e retorna um utilizador que encontra com base nos parâmetros recebidos.
/search/:word/:filter	GET	Recebe uma palavra e um *array* de filtros (i.e. nome, título, código, descrição, inclui ou exclui) e consulta a base de dados, procurando um documento que corresponda à combinação destes parâmetros.
/info/:code/:name/:description	GET	Retorna toda a informação de uma proposta inserida por um utilizador, com base no código, nome e descrição.
/info/:code	GET	Retorna toda a informação associada ao código recebido como parâmetro.
/IDinfo/:code	GET	Retorna o ID do código enviado como parâmetro.
/users/toApprove	POST	Guarda a informação do registo efetuado por um utilizador para aprovação do administrador.
/users/list	GET	Retorna todos os registos que necessitam de aprovação do administrador.
/approveUser/:id	PUT	Recebe um ID referente ao documento que contém os dados provisórios do registo do utilizador e guarda os dados na coleção dos utilizadores, indicando assim que o utilizador se encontra registado e já pode efetuar o *login*.
/admin/ searchAppUsers/:word/:filter	GET	Consulta a base de dados pelos dados dos utilizadores que ainda não se encontram registados e retorna os resultados com base nos parâmetros recebidos (palavra a pesquisar e filtro da pesquisa).
/userExists/:email/:username	GET	Verifica se o *email* e o *username* inseridos já existem na coleção dos utilizadores.
/deleteUser/:id	DELETE	Apaga os dados do registo provisório de um determinado utilizador, recebendo o ID do documento que contém tais informações.

URL	Método	Descrição
/toapprove	POST	Guarda a proposta efetuada por um utilizador para ser aprovada pelo administrador posteriormente. Verifica ainda se a proposta já existe na base de dados.
addData/:id	POST	Guarda um exemplo adicionado pelo administrador num determinado código da CIF na base de dados.
/approved/:oldId /:newId/:description/:type /:username	POST	Introduz a proposta de um exemplo submetido por um utilizador na classificação CIF, no seu respetivo código.
/rejectedProposal/:id	GET	Retorna todas as propostas que o administrador rejeitou, assim como os seus respetivos dados.
/approvedProposals	GET	Retorna todas as propostas que o administrador aprovou, assim como os seus respetivos dados.
/proposals	GET	Consulta e retorna todas as propostas efetuadas pelos utilizadores que ainda não foram aprovadas
/userProposals/:username	GET	Retorna todas as propostas submetidas por um determinado utilizador, cujo *username* é recebido como parâmetro.
/proposals/ positiveVotes/:id/:username	PUT	Adiciona um voto positivo a uma determinada proposta que se encontra por aprovar pelo administrador. Inicialmente é verificado se o utilizador já votou positivo na proposta em questão. Caso tenha votado, então não é permitida a votação. Caso seja a primeira vez que está a votar positivo no entanto, já efetuou um voto negativo, então é necessário retirar esse voto assim como o seu nome, e acrescentar um voto positivo. Por fim, se é realmente a primeira vez que está a votar na proposta é adicionado o seu voto positivo assim como o seu nome.
/proposals/ negativeVotes/:id:username	PUT	Adiciona um voto negativo a uma determinada proposta que se encontra por aprovar pelo administrador. Inicialmente é verificado se o utilizador já votou negativo na proposta em questão. Caso tenha votado, então não é permitida a votação. Caso seja a primeira vez que está a votar negativo no entanto, já efetuou um voto positivo, então é necessário retirar esse voto assim como o seu nome, e acrescentar um voto negativo. Por fim, se é realmente a primeira vez que está a votar na proposta é adicionado o seu voto negativo assim como o seu nome.
/delProposal/:id	PUT	Altera o estado da proposta com o ID recebido nos parâmetros para rejeitada.

URL	Método	Descrição
/editCIF /:incluiarray/:excluiarray/:id/:username	PUT	Esta função têm como objetivo guardar as edições, efetuadas pelo administrador, nos campos inclui e exclui. Recebe dois *arrays*, um para o campo inclui e outro para o exclui, com as respetivas edições realizadas e os dados que já existiam. Caso não se encontrem vazios são substituídos pelos já existentes na base de dados. Recebe ainda o id do documento ao qual pertence os campos editados e o *username* do administrador para guardar no histórico a edição efetuada.
/addPost	PUT	No caso do utilizador ser o administrador e pretender criar um tópico de discussão no fórum, é automaticamente adicionado com o valor *true* no campo *approved*.
/approvePost/:id	PUT	Recebe o ID do documento que pertence a um tópico de discussão criado por um utilizador e altera o campo *approved* para *true*. Permitindo assim que este apareça na página do Fórum.
/forum/posttoapprove	PUT	Cria um novo documento com os dados de um tópico de discussão criado por um utilizador, no entanto com o campo *approved* a falso, indicando que necessita da aprovação do administrador.
/forum/approvedPosts	GET	Retorna os tópicos de discussão que se encontram aprovados e abertos para discussão.
/forum/closedPosts	GET	Retorna todos os tópicos de discussão que se encontram aprovados, no entanto já concluídos.
/unapprovedPosts	GET	Retorna todos os tópicos de discussão que ainda não foram aprovados.
/delPost/:id	DELETE	Apaga o documento com o ID enviado no parâmetro, que corresponde a um tópico de discussão.
/closePost/:id	PUT	Atualiza o valor do campo *closed* para *true*, do documento referente ao ID recebido. Indicando que o tópico de discussão se encontra finalizado.
/forum/postInfo/:id	GET	Retorna toda a informação relativa ao tópico de discussão com o respetivo ID recebido.
/forum/ addComment/:id/:comment /:author	PUT	Recebe o ID do tópico de discussão, o comentário e o autor que irão ser associados na base de dados.
/searchPosts/:word/:page	GET	Consulta na base de dados um tópico de discussão já aprovado que contenha a palavra pesquisada e a categoria onde pretende pesquisar (tópicos finalizados ou em discussão).
/users/register	POST	Adiciona os dados inseridos pelo utilizador no registo à base de dados, possibilitando assim o *login*.

URL	Método	Descrição
/users/authenticate	POST	Autentica os dados inseridos pelo utilizador no *login*. Verifica se o nome de utilizador que inseriu existe, e se existe verifica se a *password* corresponde ao nome introduzido.
/users/profile	GET	Retorna as informações do utilizador que possui o *login* efetuado.
/usernameExists/:username	GET	Verifica se o *username* inserido pelo utilizador já existe na coleção *Users*.
/users/editName	PUT	Recebe o novo nome editado pelo utilizador e substitui-o pelo que existe na base de dados.
/users/editUsername	PUT	Recebe o novo *username* editado pelo utilizador e substitui-o pelo que existe na base de dados.
/users/editEmail	PUT	Recebe o novo *email* editado pelo utilizador e substitui-o pelo que existe na base de dados.
/getHistoricInclui	GET	Retorna o histórico de alterações do campo inclui, isto é, todos os códigos que possuam o campo *changes* e os seus respetivos dados relativos ao campo inclui.
/getHistoricExclui	GET	Retorna o histórico de alterações do campo exclui, isto é, todos os códigos que possuam o campo *changes* e os seus respetivos dados relativos ao campo exclui.
/searchHistoric/:word/:filter	GET	Recebe uma palavra e o filtro da pesquisa (i.e. código ou nome do utilizador). De seguida consulta na base de dados os documentos com o campo *changes*, isto é, o histórico, que correspondam a uma combinação destes dois parâmetros.

Anexo B - Descrição de cada Componente.

Na tabela seguinte, está descrito o propósito de cada Componente presente no lado do cliente.

Tabela 19 - Descrição de cada Componente.

Componentes	Descrição
add-post-forum	Para adicionar novos tópicos de discussão ao fórum.
admin	Somente autorizado ao administrador e permite verificar todas as propostas dos utilizadores que necessitam de aprovação do mesmo.
admin-detail	Somente autorizado ao administrador e permite ver os detalhes de uma proposta submetida por um utilizador e aprová-la ou rejeitá-la.
approve-posts	Somente autorizado ao administrador com o objetivo de este aprovar ou rejeitar tópicos de discussão sugeridos pelo utilizador.
approve-users	Somente autorizado ao administrador com o objetivo de este aprovar ou rejeitar os pedidos de registos efetuados pelos utilizadores.
basedata	Disponível a todo o tipo de utilizadores, mesmo aos não registados. Permite consultar e pesquisar a classificação CIF.
comment-section	Permite inserir comentários no tópico de discussão selecionado.
detail-post-forum	Permite a leitura detalhada do tópico de discussão selecionado.
edit	Somente autorizado ao administrador e utilizadores registados. Permite adicionar propostas relativas às instâncias inclui e exclui dos códigos da CIF.
edit-cif	Somente autorizado ao administrador e permite editar as instâncias inclui e exclui dos códigos da CIF.
forum	Somente autorizado ao administrador e utilizadores registados. Disponibiliza o fórum, com tópicos de discussão finalizados, em discussão e uma pesquisa sobre os mesmos.
historic	Somente autorizado ao administrador e permite averiguar e pesquisar todas as alterações realizadas à classificação CIF, nas instâncias inclui e exclui.
home	Disponível a todo o tipo de utilizadores, mesmo aos não registados. Apresenta a página inicial.
login	Possibilita o login aos utilizadores.
navbar	Contém a barra superior com determinadas restrições.
profile	Somente autorizado a utilizadores com a sessão iniciada e proporciona a verificação e edição das suas informações do utilizadores que se encontra com a sessão iniciada, assim como a possibilidade de observar o estado de todas as propostas que já submeteu.
proposals	Somente autorizado a utilizadores com a sessão iniciada e possibilita a navegação de todas as propostas aprovadas, por aprovar e rejeitadas. Além disto, possibilita a votação positiva ou negativa nas propostas por aprovar.
register	Possibilita o registo dos utilizadores.
search-users	Somente autorizado ao administrador e permite a pesquisa sobre os utilizadores registados.
sidebar-admin	Contém a barra lateral da página exclusiva ao administrador.
user-proposals	Somente autorizado a utilizadores com a sessão iniciada e proporciona a lista de todas as propostas submetidas pelo utilizador que se encontra com a sessão iniciada.

Anexo C - Descrição detalhada das funções de cada Componente

Na tabela seguinte, estão descritas em detalhe as funções relativas a cada Componente (ficheiro `componente.ts`) que constitui o lado do cliente.

Tabela 20 - Descrição das funções de cada Componente.

navbar.component	
Função	**Descrição**
onLogoutClick()	Esta função é invocada quando o utilizador clica no botão "Sair". Termina a sessão do utilizador através da chamada da função *logout()* do serviço *authService*, que apaga os dados do armazenamento local do *browser*. No fim o utilizador é redirecionado para a página do *login*.
register.component	
Função	**Descrição**
onRegisterSubmit()	Esta função é constituída por uma série de verificações sobre os campos do registo preenchidos no ficheiro *html*. Valida se o utilizador preencheu todos os campos, se o *email* e a palavra-passe introduzidos são válidos e se já existe algum utilizador com o mesmo *email* ou *username*. Caso esteja tudo correto então são enviados os dados do utilizador para o serviço *authService*.
saverange(event)	Verifica se o utilizador selecionou a opção "Outro" no campo Entidade Emissora para alterar o valor do objeto *other*. Caso esteja com valor *true*, então o campo "Número do Documento" é removido da interface do utilizador e aparece o campo "Comentário".
login.component	
Função	**Descrição**
onLoginSubmit()	Esta função é invocada quando o utilizador efetua o *login* e verifica se os campos foram todos preenchidos e se o utilizador se encontra registado. Caso esteja, os dados do *login* são guardados no armazenamento local do *browser*.
home.component	
Não contém funções no *componente.ts*, uma vez que é constituído por dados estáticos no ficheiro *html* e *links* para outras páginas.	

profile.component	
Função	**Descrição**
ngOnInit()	Quando o componente é inicializado é invocada a função *getProfile()* do serviço *authService*, com o propósito de retornar os dados do utilizador que se encontra com a sessão iniciada para serem dispostos no ficheiro *html*.
editEmail()	Função destinada à edição do *email*.
editName()	Função destinada à edição do nome.
editUsername	Função destinada à edição do *username*.
basedata.component	
Função	**Descrição**
ngOnInit()	Quando o componente é iniciado o método *getBase()* dispara um pedido HTTP GET no serviço *basedataService* para retornar os códigos do primeiro nível da CIF. Quando os dados retornam são atribuídos à variável *dataaaray*. Também é iniciado um mapa com os valores a falso, que serve para mapear as opções da pesquisa.
getlvl(code:any)	Esta função é chamada quando uma linha da tabela da página CIF é selecionada. Recebe o código que o utilizador selecionou, faz o pedido dos códigos do nível seguinte e atribui novamente à variável *dataarray* para serem dispostos na tabela.
gettitles(interval:any)	Esta função é chamada quando uma linha da tabela da página CIF é selecionada. Recebe um intervalo de códigos que o utilizador selecionou, faz o pedido dos códigos que estejam dentro desse intervalo e atribui novamente à variável *dataarray* para serem dispostos na tabela.
transform(values)	Recebe um *array* de dados e altera a ordem do mesmo.
setPage(page:number, siz:number)	Recebe o número da página e o tamanho do *array* para criar numeração das páginas.
searchData(word: any)	Recebe a palavra a ser pesquisada, verifica quais as opções do filtro da pesquisa foram selecionadas para fazer um pedido HTTP GET com a palavra e um *array* de opções. Após a consulta à base de dados, recebe os dados que correspondem à pesquisa e atribui à variável *dataarray* para serem dispostos na tabela da interface do utilizador.
updateCheckedOptions(option, event)	Esta função está a "escutar" quais as opções do filtro da pesquisa são selecionadas e guarda-as no mapa *optionsMap*.
rootClicked()	Atualiza a página atual, provocando assim a navegação para o primeiro nível da CIF.
backClicked()	Esta função permite ao utilizador regressar ao nível anterior da CIF.

edit.component	
Função	**Descrição**
ngOnInit()	Quando o componente é iniciado recolhe os dados do código que foi selecionado para a edição através do serviço *basedataService*. Estes dados foram guardados previamente em objetos dentro do serviço quando o utilizador selecionou o botão "Adicionar" do componente *basedata*.
openConfirm(e:any)	Esta função é chamada quando o utilizador submete os dados do *form* que preencheu no ficheiro *html*. Recebe-os como parâmetro e caso o utilizador que se encontra com a sessão iniciada seja o administrador, é invocada a função *approveData()* do serviço *adminService*. Se se trata de um utilizador comum, então é invocada a função t*oApprove()* do serviço *basedataService*. Em ambas as funções, é enviado um formulário com os dados e realizado um pedido HTTP POST.
edit-cif.component	
Função	**Descrição**
ngOnInit()	Quando o componente é iniciado são recuperados os dados que o administrador pretendeu editar, que se encontram no serviço *basedataService*.
incluiEdit(event, i)	Esta função recebe todas as alterações realizadas sobre o campo inclui e o índice do *array* no qual a alteração foi realizada. Tudo isto é guardado num mapa.
excluiEdit(event, i)	Esta função recebe todas as alterações realizadas sobre o campo exclui e o índice do *array* no qual a alteração foi realizada. Tudo isto é guardado num mapa.
openConfirm()	Quando o administrador finaliza as alterações e as submete, esta função é invocada. Primeiro verifica se realmente houve alguma alteração, se sim, percorre os mapas criados nas duas funções anteriores separadamente e coloca num *array* os dados não alterados, juntamente com os que foram. Por fim, envia estes *arrays* para o serviço *basedataService* através da função *editCIF*(), onde por sua vez, serão enviados para o servidor através de um método PUT.

forum.component	
Função	**Descrição**
ngOnInit()	Quando o componente é iniciado, o método *getApprovedPosts()* dispara um pedido HTTP GET no *forumService* para retornar os tópicos de discussão que se encontram aprovados e disponíveis para comentar. Quando os dados retornam são atribuídos à variável *postsData*. Também é iniciada uma variável a falso que controla se a página atual é relativa aos tópicos disponíveis para discussão ou já finalizados. Por fim, é inicializada a numeração das páginas.
openPosts()	Aquisição dos tópicos de discussão que se encontram disponíveis para discussão, novamente através da função *getApprovedPosts()*. O valor da variável *closed* é alterado para *false*.
closedPosts()	Aquisição dos tópicos de discussão que se encontram finalizados, através da invocação da função *getClosedPosts()*. O valor da variável *closed* é alterado para *true*.
transform(values)	Recebe um *array* de dados e altera a ordem do mesmo.
setPage(page:number, siz:number)	Recebe o número da página e o tamanho do *array* para criar numeração das páginas.
formattedDate(date)	Recebe um objeto com uma data e hora, retornando-o num formato convencional.
searchPost(word: any)	Recebe a palavra que o utilizador introduziu para pesquisar, envia-a para o serviço *forumService* juntamente com a variável *closed*. O resultado é constituído por um *array* de tópicos em discussão ou finalizados, consoante o valor da variável *closed* enviada, que contém a palavra pesquisada.
closePost(e: any, postID: any)	A função é accionada quando o administrador clica no botão "fechar tópico". Esta envia o ID do documento que corresponde ao tópico de discussão selecionado, para prosseguir à alteração do seu estado para finalizado.
deletePost(e: any, postID: any)	A função é accionada quando o administrador clica no botão "apagar tópico". Esta envia o ID do documento que corresponde ao tópico de discussão selecionado, para prosseguir à sua remoção.
add-post-forum.component	
Função	**Descrição**
savePost()	Esta função recebe o título e a descrição do comentário criado pelo utilizador e cria um objeto estes dados e com o nome do utilizador. Caso se trate do administrador, envia este objeto para a função *addPost*(), caso seja um utilizador comum envia para a função *postToApprove*() do serviço *forumService*, para ser guardado na base de dados

comment-section.component	
Função	**Descrição**
ngOnInit()	Quando o componente é iniciado, o método *getApprovedPosts()* dispara um pedido HTTP GET no *forumService*, para retornar os dados do tópico de discussão que foi selecionado no componente do fórum. Quando os dados retornam são atribuídos à variável *postsData*.
formattedDate(date)	Recebe um objeto com uma data e hora, retornando-o num formato convencional.
openConfirm(e: any, postId:any)	Esta função é chamada quando o utilizador submete um comentário. Recebe como parâmetro o comentário e o ID do tópico de discussão no qual foi realizado o comentário. É invocada a função *addComment*() que vai enviar um pedido PUT. Se for bem sucedido, o utilizador é redirecionado para a página do fórum.
closePost(e: any, postID: any)	A função é accionada quando o administrador clica no botão "fechar tópico". Esta envia o ID do documento pertencente ao tópico de discussão selecionado para prosseguir à alteração do seu estado para finalizado.
deletePost(e: any, postID: any)	A função é accionada quando o administrador clica no botão "apagar tópico". Esta envia o ID do documento pertencente ao tópico de discussão selecionado para prosseguir à sua remoção.
sidebar-admin.component	
Não contém funções no *componente.ts*, uma vez que é constituído por dados estáticos no ficheiro *html* e *links* para outras páginas.	
admin.component	
Função	**Descrição**
ngOnInit()	Quando o componente é iniciado, o método *getAddedData*() dispara um pedido HTTP GET no *adminService* para retornar todas as propostas realizadas por utilizadores. Quando os dados retornam são atribuídos à variável *dataarray*.
setPage(page: number, siz:number)	Recebe o número da página e o tamanho do *array* para criar numeração das páginas.
admin-detail.component	
Função	**Descrição**
ngOnInit()	Quando o componente é iniciado são recuperados os dados da proposta que o administrador selecionou no componente admin, que se encontram no serviço *adminService*.
approve(e: any, description, type, oldDataID, newDataId, username)	Caso o administrador clique no botão "aprovar" apresentado no *html*, esta função é invocada e são recolhidas todas as informações relativas à proposta, que depois são enviadas para a função *approveData*() do serviço *adminService*, para serem introduzidas na base de dados.
delete(e: any, newDataId)	Caso o administrador clique no botão "apagar" apresentado no *html*, esta função é invocada e a proposta é removida da base de dados, através de um pedido HTTP DELETE contido na função *delApproved*() do serviço *adminService*.

approve-users.component	
Função	**Descrição**
ngOnInit()	Quando o componente é iniciado são requisitados os dados de todos os utilizadores que necessitam de aprovação do administrador, para poderem concluir o registo. Também é inicializada a numeração das páginas.
setPage(page: number, siz:number)	Recebe o número da página e o tamanho do *array* para criar numeração das páginas.
approve(e: any, userID)	Caso o administrador clique no botão "aprovar" apresentado no *html*, esta função é invocada e é enviado o ID do utilizador a registar para a função *approveUser*() do serviço *adminService* para ser introduzido na base de dados como um utilizador. De seguida os dados provisórios do pedido de registo são apagados.
reject(e: any, userID)	Caso o administrador clique no botão "rejeitar" apresentado no *html*, esta função é invocada e é enviado o ID do utilizador, que fez o pedido de registo, para a função *delUser*() do serviço *adminService* para que os dados do pedido de registo sejam apagados.
searchUser(word: any, filter:any)	Recebe a palavra que o administrador introduziu para pesquisar um utilizador e um filtro (nome, *username*, *email* ou número do documento) que são enviados para o serviço *adminService*. O resultado é constituído por um *array* de pedidos de registo que correspondem com a pesquisa efetuada.
search-users.component	
Função	**Descrição**
ngOnInit()	Quando o componente é iniciado são requisitados os dados de todos os utilizadores registados. Também é inicializada a numeração das páginas.
setPage(page: number, siz:number)	Recebe o número da página e o tamanho do *array* para criar numeração das páginas.
searchUser(word: any, filter:any)	Recebe a palavra que o administrador introduziu para pesquisar um utilizador e um filtro (nome, *username*, *email* ou número do documento) que são enviados para o serviço *adminService*. O resultado é constituído por um *array* utilizadores registados que correspondem com a pesquisa efetuada.

approve-posts.component	
Função	**Descrição**
ngOnInit()	.Quando o componente é iniciado, o método *getPosts()* dispara um pedido HTTP GET no *adminService* para retornar todas as propostas de tópicos de discussão do fórum. Quando os dados retornam são atribuídos à variável *postsData*.
setPage(page: number, siz:number)	Recebe o número da página e o tamanho do *array* para criar numeração das páginas.
formattedDate(date)	Recebe um objeto com uma data e hora, retornando-o num formato convencional.
transform (values)	Recebe um *array* de dados e altera a ordem do mesmo.
approve(e: any, postID)	Caso o administrador clique no botão "aprovar" apresentado no *html*, esta função é invocada e é enviado o ID do tópico de discussão a aprovar para a função *approvePostAdmin()* do serviço *adminService* para ser introduzido na base de dados e aparecer na página do Fórum.
reject(e: any, postID)	Caso o administrador clique no botão "rejeitar" apresentado no *html*, esta função é invocada e é enviado o ID do tópico de discussão para a função *delPost()* do serviço *adminService* para seja apagado.
historic.component	
Função	**Descrição**
ngOnInit()	.Quando o componente é iniciado, o método *getHistoric()* dispara um pedido HTTP GET no *adminService* para retornar um *array* com documentos que tenham sofrido alterações.
formattedDate(date)	Recebe um objeto com uma data e hora, retornando-o num formato convencional.
transform (values)	Recebe um *array* de dados e altera a ordem do mesmo.
searchhistoric(word: any, filter:any)	Recebe a palavra que o administrador introduziu para realizar uma pesquisa sobre o histórico e um filtro (utilizador ou código) que são enviados para o serviço *adminService*. O resultado é constituído por um *array* de documentos que correspondam com a pesquisa efetuada.
inclui()	Quando este método é invocado, a função *getHistoricInclui()* dispara um pedido HTTP GET no *adminService* para retornar todas as alterações realizadas sobre os campos inclui da CIF. Quando os dados retornam são atribuídos à variável *historic.*
exclui()	Quando este método é invocado, a função *getHistoricInclui()* dispara um pedido HTTP GET no *adminService* para retornar todas as alterações realizadas sobre os campos inclui da CIF. Quando os dados retornam são atribuídos à variável *historic.*

proposals.component	
Função	**Descrição**
ngOnInit()	.Quando o componente é iniciado, são invocados três métodos do serviço *proposalsService: toApprove(), rejectedProposals()* e *approvedProposals()*. Estas funções disparam pedidos HTTP GET para retornar as propostas que se encontram por aprovar, rejeitadas e aprovadas, respetivamente.
setPage(page: number, siz:number)	Recebe o número da página e o tamanho do *array* para criar numeração das páginas.
formattedDate(date)	Recebe um objeto com uma data e hora, retornando-o num formato convencional.
transform (values)	Recebe um *array* de dados e altera a ordem do mesmo.
getToApprove()	Retorna os dados das propostas que estão por aprovar para a página HTML e altera a secção na barra horizontal.
getRejected()	Retorna os dados das propostas que se encontram rejeitadas para a página HTML e altera a secção na barra horizontal.
getApproved()	Retorna os dados das propostas que estão aprovadas para a página HTML e altera a secção na barra horizontal.
positiveVote(id)	Recebe o ID da proposta que irá receber um voto positivo. Invoca o método *positiveVotes()* do serviço *proposalsService* e envia-lhe o *id* e o *username* do utilizador que realizou o voto.
negativeVote(id)	Recebe o *ID* da proposta que irá receber um voto negativo. Invoca o método *negativeVotes()* do serviço *proposalsService* e envia-lhe o *id* e o *username* do utilizador que realizou o voto.
user-proposals.component	
Função	**Descrição**
ngOnInit()	.Quando o componente é iniciado, o método *getUserProposals()* dispara um pedido HTTP GET no *proposalsService* para retornar um *array* com todas as propostas submetidas pelo utilizador que está atualmente com a sessão iniciada.
setPage(page: number, siz:number)	Recebe o número da página e o tamanho do *array* para criar numeração das páginas.
formattedDate(date)	Recebe um objeto com uma data e hora, retornando-o num formato convencional.

Anexo D - Descrição detalhada das funções de cada Serviço

Na tabela seguinte, são descritas em detalhe as funções relativas a cada Serviço que constitui o *cliente side*.

Tabela 21 - Descrição das funções de cada Serviço.

auth.service	
Funções	**Descrição**
userToApprove(user)	Pedido HTTP POST para guardar os dados do registo do utilizador para serem aprovados.
registerUser(user)	Pedido HTTP POST para registar o utilizador enviado no URI.
isAdmin()	Verifica se o utilizador com a sessão iniciada é o administrador.
userExists(email, username)	Pedido HTTP GET para verificar se o utilizador (*email* e *username*) já existe na base de dados.
usernameExists(username)	Pedido HTTP GET para verificar se o utilizador (*username*) já existe na base de dados.
editEmail(user)	Pedido HTTP PUT para substituir o *email* recebido por aquele que se encontra na base dados.
editUsername(user)	Pedido HTTP PUT para substituir o *username* recebido por aquele que se encontra na base dados.
editName(user)	Pedido HTTP PUT para substituir o nome recebido por aquele que se encontra na base dados.
authenticateUser(user)	Pedido HTTP POST para autenticar as credenciais do utilizador aquando o *login*.
getProfile()	Pedido HTTP GET que retorna todas as informações do utilizador.
getLoginName()	Retorna o nome do utilizador que se encontra com a sessão iniciada no momento.
storeUserData(token, user)	Guarda a informação do utilizador no armazenamento local do *browser*.
loadToken()	Carrega o *token* do armazenamento local.
loggedIn()	Indica se o utilizador está com a sessão iniciada de momento.
logout()	Apaga os dados do utilizador da sessão, ou seja, termina a sessão do mesmo.
validate.service	
Funções	**Descrição**
validateRegister(user)	Verifica se os campos da página do registo são válidos.
validateLogin(user)	Verifica se os campos da página do *login* são válidos.
validateEmail(email)	Verifica se o *email* é válido.
validatePassword(password, passwordrepeat)	Valida se as duas palavras-passe introduzidas no registo são iguais.

admin.service	
Funções	**Descrição**
getAddedData()	Pedido HTTP GET que retorna todas as propostas dos utilizadores
setDetails(codeAttached, userName,userDescription)	Guarda as informações referentes a uma proposta para serem utilizadas por outro componente.
getOneData()	Pedido HTTP GET que retorna toda a informação relativa a uma proposta inserida por um utilizador.
search_approvedUsers(word, filter)	Pedido HTTP GET que retorna as informações referentes a utilizadores ainda não aprovados, de acordo com os dados inseridos na pesquisa (palavra e o filtro).
searchInfoCode()	Pedido HTTP GET que retorna toda a informação correspondente ao código enviado no URI.
searchInfoCode2(code)	Pedido HTTP GET que retorna o ID do código enviado no URI.
getRecords()	Pedido HTTP GET que retorna todos os registos ainda não aprovados, efetuados por utilizadores.
getAllUsers()	Pedido HTTP GET que retorna os dados de todos utilizadores registados na plataforma.
getUser(word, filter)	Pedido HTTP GET que retorna toda a informação referente a utilizadores, que correspondam com os dados submetidos na pesquisa.
getHistoricInclui()	Pedido HTTP GET que retorna o histórico das alterações efetuadas sobre a instância inclui de um código.
getHistoricExclui()	Pedido HTTP GET que retorna o histórico das alterações efetuadas sobre a instância exclui de um código.
searchHistoric(word, filter)	Pedido HTTP GET que retorna dados referentes ao histórico que correspondam com os objetos inseridos na pesquisa.
approveUser(userID)	Pedido HTTP PUT que aprova e regista o utilizador referido no URI.
delUser(userID)	Pedido HTTP DELETE que remove o registo do utilizador referido no URI.
addPost(post)	Pedido HTTP PUT que cria um novo tópico de discussão.
approvePostAdmin(postId)	Pedido HTTP PUT que aprova o tópico de discussão mencionado no URI.
getPosts()	Pedido HTTP GET que retorna todos os tópicos de discussão ainda não aprovados pelo administrador.
approveData(oldId, newId,description,type, username)	Pedido HTTP PUT que guarda a proposta realizada pelo utilizador na base de dados.
addData(form, id)	Pedido HTTP POST que guarda automaticamente a proposta realizada pelo administrador na base de dados.
delPost(postId)	Pedido HTTP DELETE que remove um tópico de discussão da base de dados.
closePost(postId)	Pedido HTTP PUT que finaliza o tópico de discussão mencionado no URI.
delApproved(newDataId)	Pedido HTTP DELETE que remove a proposta do utilizador da base de dados.

basedata.service	
Funções	**Descrição**
getBase()	Pedido HTTP GET que retorna os códigos do primeiro nível da CIF (***b,s,d,e***).
getlvl(code)	Pedido HTTP GET que retorna os códigos do nível seguinte, relativos ao código mencionado no URI.
gettitles(code)	Pedido HTTP GET que recebe um intervalo de códigos e retorna aqueles que se encontram dentro desse mesmo intervalo.
search(word, filter)	Pedido HTTP GET que retorna os dados que correspondem à pesquisa efetuada (palavra ou código e o filtro).
searchInfoCode(code)	Pedido HTTP GET que retorna toda a informação referente ao código recebido.
editCIF(incluiarray, excluiarray, id,username)	Pedido HTTP PUT para guardar todas as edições realizadas sobre as instâncias inclui e exclui de um determinado código e o utilizador que as efetuou.
editData(data)	Guarda os dados de um determinado código para ser utilizado por outro componente.
getEditedData()	Retorna os dados guardados no serviço em questão.
getCode()	Retorna o código guardado no serviço em questão.
toApprove(form)	Pedido HTTP POST para guardar os dados relativos a uma proposta de um utilizador para serem posteriormente aprovados pelo administrador.
forum.service	
Funções	**Descrição**
postToApprove(post)	Pedido HTTP PUT para guardar a informação do tópico de discussão criado por um utilizador para ser posteriormente aprovado.
getApprovedPosts()	Pedido HTTP GET que retorna todos os tópicos de discussão aprovados e disponíveis para discussão.
getClosedPosts()	Pedido HTTP GET que retorna todos os tópicos de discussão aprovados e finalizados.
commentPage(postId)	Guarda o ID do tópico de discussão para ser utilizado por outro componente.
getID()	Retorna o ID do tópico de discussão guardado no serviço em questão.
getPostInfo()	Pedido HTTP GET que retorna toda a informação do tópico de discussão referente ao ID enviado no URI.
search_posts(word, page)	Pedido HTTP GET que retorna todos os tópicos de discussão que correspondem aos dados pesquisados pelo utilizador(palavra e filtro)
addComment(postId, comment, author)	Pedido HTTP PUT que guarda o comentário realizado no tópico de discussão mencionado e o autor que o efetuou.

proposals.service	
Funções	**Descrição**
toApprove()	Pedido HTTP GET para retornar todas as propostas submetidas que se encontram por aprovar.
rejectedProposals()	Pedido HTTP GET para retornar todas as propostas submetidas que se encontram por rejeitadas.
approvedProposals()	Pedido HTTP GET para retornar todas as propostas submetidas que se encontram aprovadas.
positiveVotes(id, username)	Pedido HTTP PUT para colocar um voto positivo numa determinada proposta.
negativeVotes(id, username)	Pedido HTTP PUT para colocar um voto negativo numa determinada proposta.
getUserProposals(username)	Pedido HTTP GET que retorna todas as propostas submetidas por um determinado utilizador.

Anexo E - Descrição dos Guardas.

Na tabela seguinte, estão descritos os objetivos de cada Guarda presente no lado do cliente.

Tabela 22 - Descrição das Guardas.

Guardas	Descrição
adminnavg.guards.ts	Impede que o administrador aceda à rota "*/management/detail*" diretamente, pois apenas é possível aceder a estas através da rota "*/managment*".
auth.guards.ts	Impede que o utilizador aceda a determinadas páginas se não tiver efetuado o *login*.
databasenavg.guards.ts	Impede que o utilizador aceda às rotas "*/base/edit*" e *"base/editCIF*" diretamente, pois apenas é possível aceder a esta através da rota "*/base*".
navigation.guards.ts	Impede que o administrador aceda à rota "*/forum/commentSection*" diretamente, pois apenas é possível aceder a esta através da rota "*/forum*".
role.guards.ts	Protege as rotas que apenas são destinadas ao administrador.